STADIA
HELSINGIN AMMATTIKORKEAKOULU

TEKNIIKAN JA LIIKENTEEN TOIMIALA

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

Java 3D

Työn tekijä: Samuli Elomaa
Työn valvoja: Matti Luukkainen
Työn ohjaaja: Matti Luukkainen

Työ hyväksytty: __. __. 2008

Matti Luukkainen
lehtori

ALKULAUSE

Tämä insinöörityö tehtiin Helsingin Ammattikorkeakoulu Stadian tekniikan ja liikenteen yksikölle. Kiitän perhettäni ja insinöörityöni ohjaajaa.

Vantaalla 17.04.2008

Samuli Elomaa

STADIA
HELSINGIN AMMATTIKORKEAKOULU

INSINÖÖRITYÖN TIIVISTELMÄ

Tekijä: Samuli Elomaa	
Työn nimi: Java 3D	
Päivämäärä: 17.04.2008	Sivumäärä: 51
Koulutusohjelma: Tietotekniikka	Suuntautumisvaihtoehto: Ohjelmistotekniikka
Työn valvoja: lehtori Matti Luukkainen Työn ohjaaja: Matti Luukkainen	

Tämä insinöörityö kertoo Java 3D -ohjelmointirajapinnan perusteista ja sen käytöstä kolmiulotteisen tietokonegrafiikan luomisessa Java ohjelmointikielellä. Java 3D on rajapinta Java-ohjelmointikielelle, jonka avulla voidaan luoda ja käsitellä kolmiulotteista tietokonegrafiikkaa. Java 3D -rajapinnan avulla käsitellään kolmiulotteista tietokonegrafiikka erityisen maisemagraafimallin avulla. Maisemagraafi on binääripuuta muistuttava malli, joka mahdollistaa kolmiulotteisten kohteiden ja niille tapahtuvien muunnoksien käsittelyn hierarkisessa järjestyksessä.

Työssä käydään läpi Java 3D -maisemagraafien luominen ja kolmiulotteisessa avaruudessa sijaitseville kappaleille tehtäviä perusoperaatioita kuten siirtoa ja kiertoa. Lisäksi käydään läpi myös erilaisia animoinnissa ja interaktiossa käytettäviä luokkia, joiden avulla ohjelmoija saa automatisoitua muunnoksia sekä käyttäjä voi antaa syötteitä hiirellä ja näppäimistöllä.

Näiden lisäksi käydään läpi myös mallin valaistusta, varjoja, teksturointia sekä omien kolmiulotteisten mallien tuontia Java 3D -maailmaan. Opinnäytetyön yhteydessä on tehty myös kirjo erilaisia esimerkkejä, jotka ovat saatavilla verkkosivustolta osoitteessa http://www.pahvilaatikko.org/j3d lisäksi kopio verkkosivustosta löytyy myös opinnäytetyön mukana tulevalta cd-levyltä.

Avainsanat: Java 3D, Java, OpenGL, Tietokonegrafiikka

STADIA
HELSINGIN AMMATTIKORKEAKOULU

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ABSTRACT

Name: Samuli Elomaa

Title: Java 3D

Date: 17.04.2008 Number of pages: 51

Department: Tekniikka ja Liikenne Study Programme: Software Engineering

Instructor: Matti Luukkainen

Supervisor: Matti Luukkainen

This engineering thesis is about Java 3D API basics and usage for creating 3D computer graphics with Java programming language. Java 3D is an Application Programming Interface library for Java programming language. Java 3D is based on scene graph model. Scene graph model is a binary-tree like hierarchy where 3D objects and their transforms are handled in a hierarchical way.

Thesis goes through Java 3D scene graph models, basic 3D transforms such as moving or rotating objects, animation and interaction by keyboard and mice. Also lights, shadows, textures, and importing user created 3D models to Java 3D world are walked through with examples.

Examples for this thesis are available at web-address http://www.pahvilaatikko.org/j3d and offline-copy of the webpages is available in the included CD-ROM.

Keywords: Java 3D, Java, OpenGL, Computer Graphics

SISÄLLYS

1 JOHDANTO 1

2 MAISEMAGRAAFI 3

3 KOLMIULOTTEINEN AVARUUS 8

3.1 Kierto ... 8
3.2 Siirto ... 13

4 VALAISTUSMALLIT JA SÄVYTYS 15

4.1 Valoitusmalliesimerkki ... 18

5 VARJOT 19

6 ANIMOINTI 21

6.1 Automaattinen kappaleen pyörittäminen RotationInterpolator-luokalla ... 21
6.2 Automaattinen siirto käyttäen PositionInterpolator-luokkaa ... 24
6.3 Automaattinen polunmuodostaminen PositionPathInterpolator-luokalla ... 26
6.4 Automaattinen kierto-sijainti-polku käyttäen RotationPositionPathInterpolator-luokkaa ... 27

7 INTERAKTIO 31

7.1 Hiiri ... 33
7.2 Näppäinnavigointi ... 35

8 TEKSTUROINTI 36

8.1 Mipmap ... 37
8.2 Mainostaulutekniikka ... 39
8.3 Detaljitason asteittainen tarkentaminen ... 41

9 OMIEN MALLIEN TUONTI JAVA 3D -MAISEMAGRAAFIIN. 44

10 YHTEENVETO 47

VIITELUETTELO 48

1 JOHDANTO

Java 3D on GPL-lisensoitu ohjelmointirajapinta sekä laajennuskirjasto, joka tarjoaa korkean tason oliopohjaisen ohjelmointiliittymän Java-virtuaalikoneen alla oleviin käyttöjärjestelmän omiin grafiikkarajapintoihin kuten DirectX:ään tai OpenGL:ään. Toisin kuin esimerkiksi proseduraalinen OpenGL-rajapinta, Java 3D –rajapinta tarjoaa Java-ohjelmointikielelle paremmin sopivan korkean tason oliopohjaisen ohjelmointiympäristön.

Siinä missä OpenGL perustuu välittömään piirtämiseen eli siihen, että grafiikkaa piirretään samalla, kun rajapinnalle syötetään käskyjä. Java 3D:ssä grafiikka muodostetaan tekemällä ns. maisemagraafi (scene-graph) -malli, joka sitten syötetään valmiina rajapinalle piirrettäväksi. Kun malli on kertaalleen rakennettu, rajapinnalle ei enää tarvitse syöttää uudelleen koko mallia, vaan mallin muokkaus voidaan toteuttaa muokkaamalla vain pientä osaa mallista. Tosin Java 3D -rajapintaa voidaan myös käyttää välittömään piirtämiseen samalla tavalla kuin OpenGL -rajapintaa.

Muita vastaavia maisemagraafin tukeutuvia rajapintoja ovat mm. VRML-kieli, SGI:n Open Inventor ja AutoCAD. Java 3D tuli ensimmäisen kerran julkisuuteen joulukuussa 1996, kun Silicon Graphics ja Sun Microsystems yhtiöt ilmoittivat kehittävänsä 3D-rajapintaa Java-ohjelmointikielelle. Myöhemmin projektiin liittyivät myös Apple ja Intel.

Ensimmäinen julkinen beta-versio julkaistiin maaliskuussa 1998, ja valmis versio 1.0 joulukuussa 1998. Vuosina 2003-2004 kehitystyö oli jäissä, kunnes Java 3D:n lähdekoodit julkaistiin Java Community Process -projektin alaisuudessa koodinimellä JSR926 kesällä 2004. Tästä lähtien kehitys on jatkunut Sunin oman kehitysryhmän lisäksi talkoovoimin, ja versio 1.4 julkaistiin helmikuussa 2006. Kirjoitushetkellä uusin versio rajapinnasta versio 1.5.1 julkaistiin kesällä 2007. [1]

Rajapintaa käytetään erityisesti kolmiulotteiseen mallinnukseen ja simulointeihin. Käyttäjiä ovat mm. yhdysvaltalainen avaruusjärjestö NASA, Yhdysvaltain ilmavoimat, IBM ja Brighamin naistensairaala. [2]

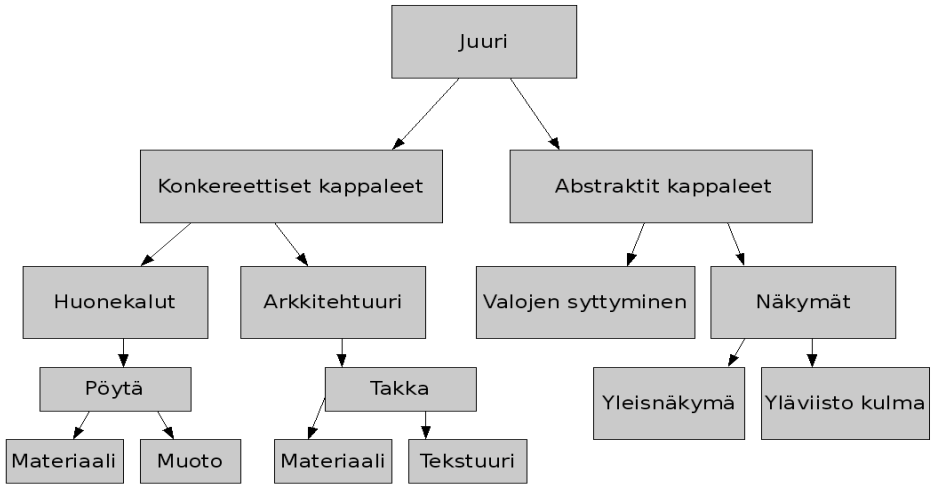
Suurin ongelma Java 3D -rajapinnan yleistymisessä on ollut sen puuttuminen Sunin Java Runtime Enviroment -paketista. Tästä johtuen käyttäjien ja kehittäjien täytyy erikseen asentaa ohjelmakirjastot, mikä on kankeaa ja vaivalloista keskivertokäyttäjää ajatellen. Lisäksi tietokoneiden näytönohjainten alkeellisuus on hidastanut Java 3D:n leviämistä. Tulevaisuudessa voitaneen odottaa Java 3D -kirjaston liittämistä kiinteästi osaksi Java-peruspakettia.

Java 3D tukee myös 3D-simuloinnissa käytettävää laitteistoa kuten virtuaalilaseja, -hanskoja sekä -kypäriä. Tämä mahdollistaa rajapinnan käytön vaativissa virtuaaliympäristöjen rakentamisissa, joissa täytyisi muutoin ohjelmoida jokaiselle laitteelle ajurit erikseen. Java 3D on siis yli 12 vuoden ohjelmistokehittämisen tulos. Kun kunnolliset kolmiulotteisen grafiikanpiirtämiseen sopivat näytönohjaimet alkavat olla arkipäivää lähes joka tietokoneessa, alkaa mahdollisesti myös kolmiulotteiselle tietokonegrafiikalle olla enemmän käyttöä keskivertotietokoneen ohjelmistoissa.

Koska Java 3D on nimenomaan rajapinta Java-ohjelmointikielelle, voi sillä tehtyjä ohjelmia ajaa kaikkialla, missä Java virtuaalikone on käytettävissä. Täten esimerkiksi kolmiulotteinen tietokonepeli toimii samoin niin Windows-, Linux- kuin Macintosh-ympäristöissä.

2 MAISEMAGRAAFI

Java 3D -rajapintaa pidetään neljännen sukupolven 3D-rajapintana. Neljännen sukupolven rajapinnat perustuvat ”maisemagraafi”-arkkitehtuuriin, jossa kolmiulotteisia kappaleita käsitellään binääripuumaisessa rakenteessa. [3] Maisemagraafi koostuu erilaisista ryhmä- ja lehtisolmuista sekä rajauslaatikoista, joiden avulla kolmiulotteista hierarkiaa käsitellään. Itse kappaleille tehtävät operaatiot ilmaistaan matriisipohjaisina affiinimuunnoksina kuten OpenGL:ssäkin.

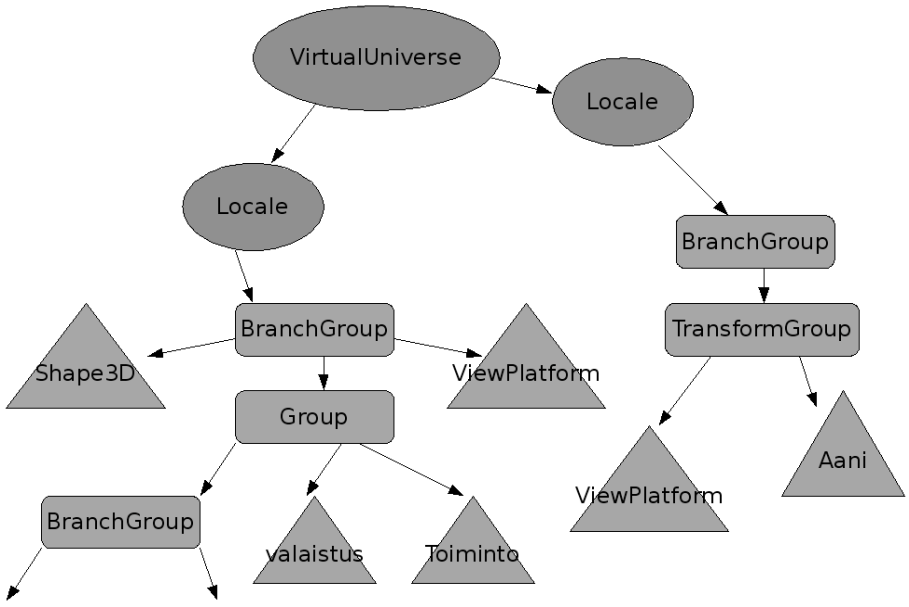


Kuva 1 Maisemagraafi-hierarkia

Maisemagraafi-hierarkiassa (kuva 1) kaikki alkaa juuresta, joka on jakautunut kahteen ryhmäsolmuun (haaraan). Nämä puolestaan ovat jakautuneet useampaan haaraan, jotka jakautuvat lopulta omiksi lehtisolmuiksi. Esimerkiksi, ”Abstraktit kappaleet” on haara, jolla on lehti ”Valojen syttyminen” sekä haara ”Näkymät”, jolla on lehdet ”Yleisnäkymä” ja ”Yläviisto kulma”. Näin ollen maisemagraafia käydään läpi rekursiivisesti solmu solmulta läpi. Tällainen rakenne on ohjelmoijan helppo hahmottaa, ja sitä on myös ohjelmallisesti helppo käsitellä. Kun tällaista rakennetta piirretään tietokoneen ruudulle, se tapahtuu käymällä läpi puuta ja piirtäen kappale kerrallaan siinä järjestyksessä kuin se esiintyy. Lainaus Java 3D –rajapinnan dokumentaatiosta:

> ”A leaf node's state is defined by the nodes in a direct path between the scene graph's root and the leaf. Because a leaf's graphics context relies only on a linear path between the root and that node, the Java 3D renderer can decide to traverse the scene graph in whatever order it wishes. It can traverse the scene graph from left to right and top to bottom, in level order from right to left, or even in parallel.” [4]

Java 3D:ssä kuvan 1 kaavio menisi siten, että maisemagraafin juurena toimii VirtualUniverse-luokka, mistä haarautuvat ryhmäsolmut (BranchGroup), ja niistä edelleen mahdolliset muunnossolmut (TransformGroup), joiden alla ovat sitten itse Shape3D-luokkaan pohjautuvat kolmiulotteiset kappaleet.



Kuva 2 Java 3D -luokkahierarkia

Hello World

Ensimmäiseksi käydään läpi perus 3D-Hello World- vastine, kolmiulotteisen kuution. Esimerkin täydellinen lähdekoodi löytyy viitteestä [5]. Kuten aiemmin todettu, maisemagraafihierarkiassa kaikki alkaa juuresta joten aloitamme luomalla instanssin SimpleUniverse-luokasta, joka on yksinkertaistettu toteutus VirtualUniverse-luokasta.

SimpleUniverse

SimpleUniverse-luokka on Universe-luokan toteutus, jonka avulla ohjelmoijan on helppo luoda juuri maisemagraafiinsa. Se luo automaattisesti kaikki maisemagraafin katseluhaarassa tarvittavat oliot. Näin ollen käyttäjän ei tarvitse itse määritellä katseluhaaraa, vaan hän voi keskittyä nopeaan sisällönluomiseen.

Ote SimpleUniverse-luokan dokumentaatiosta:

> ”This class sets up a minimal user environment to quickly and easily get a Java 3D program up and running. This utility class creates all the necessary objects on the "view" side of the scene graph. Specifically, this class creates a locale, a single ViewingPlatform, and a Viewer object (both with their default values). Many basic Java 3D applications will find that SimpleUniverse provides all necessary functionality needed by their applications. More sophisticated applications may find that they need more control in order to get extra functionality and will not be able to use this class.” [6]

SimpleUniverse-luokka luo myös automaattisesti Locale-nimisen luokan instanssin.

Locale

Locale-luokka on Java 3D –luokka, joka toimii eräänlaisena purkkina BranchGroup-luokan oliolle, ja määrittelee niiden sijainnit universumissa. Locale-oliota voi olla useita yhdessä universumissa, ja Universe-luokan tehtävänä on sitten määritellä, missä Locale-oliossa ollaan.

Ote Locale-luokan dokumentaatiosta:

> ”A Locale object defines a high-resolution position within a VirtualUniverse, and serves as a container for a collection of BranchGroup-rooted subgraphs (branch graphs), at that position. Objects within a Locale are defined using standard double-precision coordinates, relative to the origin of the Locale. This origin defines the Virtual World coordinate system for that Locale.” [7]

Tämän jälkeen luomme yhden haaran (BranchGroup) sekä kutsumalla BranchGroup-luokan metodia add. Lisäämme sille lehtisolmun, jossa on instanssi kolmiulotteisesta kuutiosta. Tulee huomata, että esimerkissä maisemagraafi alkaa oikeastaan vasta BranchGroup-haarasta. Täten yhdessä universumissa voi olla useita maisemagraafeja.

BranchGroup

Ote BranchGroup-luokan dokumentaatiosta:

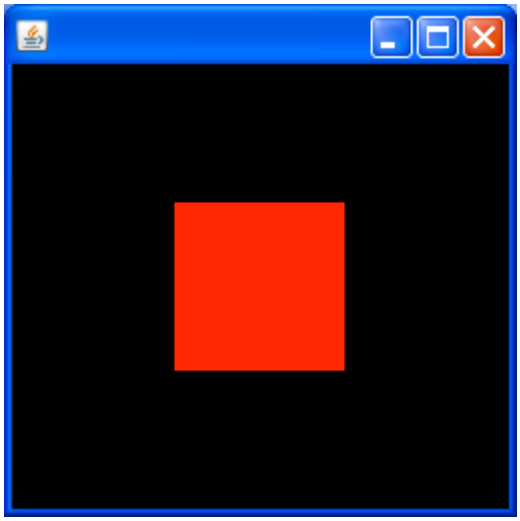
> ”The BranchGroup serves as a pointer to the root of a scene graph branch; BranchGroup objects are the only objects that can be inserted into a Locale's set of objects.” [8]

Lopuksi kutsumme itse SimpleUniverse-luokan metodia add, jolla lisäämme haaran universumiimme sekä määrittelemme katselukulman käyttäen setNominalViewingTransform-metodia.

```
SimpleUniverse universe = new SimpleUniverse();
BranchGroup group = new BranchGroup();
group.addChild(new ColorCube(0.3));
universe.getViewingPlatform().setNominalViewingTransform();
universe.addBranchGraph(group);
```

Näin olemme luoneet ensimmäisen Java 3D -luokkamme, joka ajettaessa näyttää kolmiulotteisen kuution mustalle taustalle kuvassa kolme. Tosin kuutio on kolmiulotteinen vain tietokoneen muistissa, sillä kun se projisoidaan kaksiulotteiselle näytölle, näemme vain punaisen neliön mustalla taustalla.

Katso täydellinen lähdekoodi liitteestä [5].



Kuva 3 Hello World

3 KOLMIULOTTEINEN AVARUUS

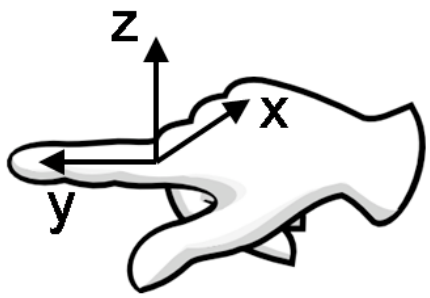
3.1 Kierto

Java 3D käyttää ns. ”oikean käden” koordinaatistojärjestelmää, jossa pisteiden x, y, z -arvot kertovat sijainnin avaruuden koordinaatistossa. Pisteet x,y,z ovat määritelty 256-bittisinä liukulukuina, jossa arvo 1.0 vastaa etäisyyttä yksi metri. Näin on mahdollista määritellä etäisyydet universumitasolta aina protonitasolle asti. Taulukossa 1 on lueteltuna Java 3D -maailmassa käytettäviä perusetäisyyksiä.

Taulukko 1 Java 3D -etäisyydet

2^n metriä	Vastaava yksikkö
87.29	Universumi (20 miljardia valovuotta)
69.68	Galaksi (100 miljoonaa valovuotta)
53.07	Valovuosi
43.43	Aurinkokuntamme ympärys
23.60	Maapallon ympärys
9.97	Kilometri
0.00	Metri
-19.93	Mikrometri
-33.22	Ångström
-115.57	Plank

Kuvassa 4 esitellään oikean käden koordinaatistojärjestelmä.



Kuva 4 Oikean käden koordinaattijärjestelmä [2]

Jotta pystyisimme näkemään kappaleen kolmiulotteisuuden, täytyy meidän muuttaa kappaleen kulmaa (katsojaa kohdin) avaruudessa x- ja y-akselin suhteen. Kierto lasketaan käyttämällä sovellettua Eulerin kaavaa, missä symboli Θ ilmoittaa kulmaa, sekä W tarkoittaa suoraa origon kautta kulkevaa suuntavektoria.

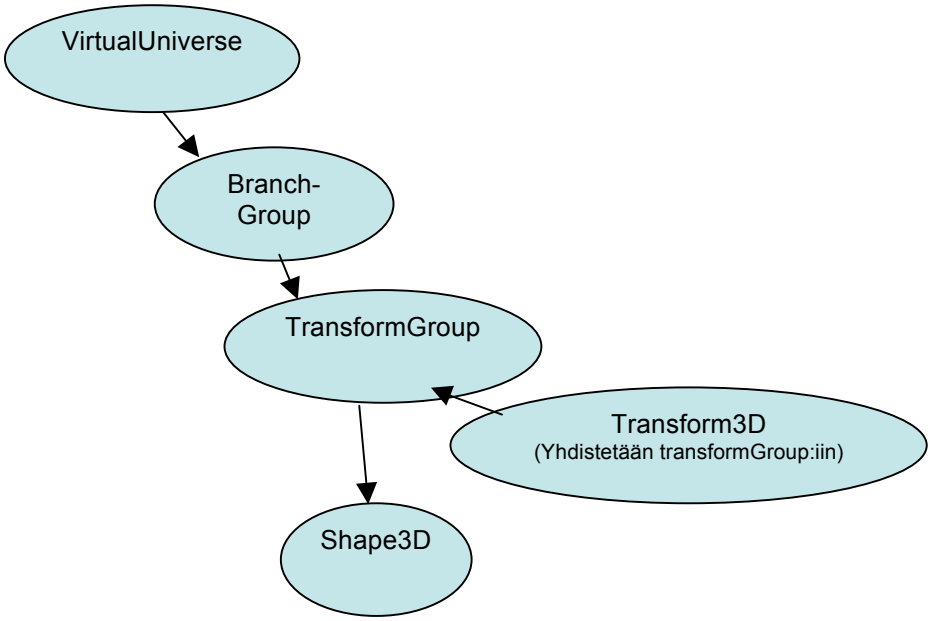
$$X' = [I + W * \sin \Theta + W^2 * (1 - \cos \Theta)] * X$$

Toteutamme muunnokset x- ja y-akselin suhteen muokkaamalla esimerkkiä luomalla BranchGroup-haaraan uuden haaran käyttämällä Java 3D:n tarjoamaa TransformGroup-luokkaa. Transform-luokat ovat vastuussa mallille syötettävistä affiinimuunnoksista (siirto, kääntö, skaalaus).

On huomattava, että TransformGroup-olio vaikuttaa kaikkiin sen haarassa alla oleviin kappaleisiin.

Ote TransformGroup-luokan dokumentaatiosta:

> ”Group node that contains a transform. The TransformGroup node specifies a single spatial transformation, via a Transform3D object, that can position, orient, and scale all of its children. The specified transformation must be affine.” [9]



Kuva 5 Ryhmähierarkia

Aloitamme lisäämällä TransformGroup-haaran:

TransformGroup muunnoshaara = new TransformGroup();

Tarvitsemme myös Transform3D-luokkaa. Transform3D-luokka ilmaisee kappaleelle tehtävät muunnokset. Teknisesti katsoen se on itseasiassa 4x4-matriisi. Huomioitavaa on, että itse Transform3D-olio ei kuitenkaan esiinny itse maisemagraafimallissa, vaan se annetaan parametrina TransformGroup-luokan metodille setTransform, joka sitten hoitaa varsinaisen affiinimuunnoksen.

Ote Transform3D-luokan dokumentaatiosta:

> A generalized transform object represented internally as a 4x4 double-precision floating point matrix. The mathematical representation is row major, as in traditional matrix mathematics. A Transform3D is used to perform translations, rotations, and scaling and shear effects.[10]

Affiinimuunnos on 4x4-matriisi, joka sisältää kaikki muunnokset yhtenä matriisina.

Esimerkiksi alla on esitetty affiinimatriisi M, jossa on yhdistetty kierto avaruuden akselin K ympäri sekä siirto avaruuden vektorin V suhteen. Lisäksi 4x4-affiinimatriisi sisältää alimmilla riveillä ylimääräiset nollat. Kaikkein oikeammalla oleva 1 kertoo, että V on vektori. Jos V olisi avaruuden piste, olisi sen kohdalla 0.

0	Kz	Ky	Vx
Kz	0	Kx	Vy
Ky	Kx	0	Vz
0	0	0	1

Tästä saadaan tarvittaessa myös 3x3-matriisi ottamalla alin rivi sekä oikeanpuoleinen siirtoa ilmaiseva rivi pois.

Esimerkissämme tarkoituksenamme on saada kappale kääntymään. Siihen tarvitaan kaksi kappaletta Transform3D-olioita, yksi kummallekin akselille. •

```
Transform3D kaannosXsuhteen = new Transform3D();
Transform3D kaannosYsuhteen = new Transform3D();
```

Määritämme näille käännöksen 45-astetta X-akselin suhteen sekä 30 astetta Y-akselin suhteen (radiaaneina).

```
kaannosXsuhteen.rotX(Math.PI/4.0d);
kaannosYsuhteen.rotY(Math.PI/6.0d);
```

Seuraavaksi meidän täytyy kertoa saadut matriisit keskenään, jolloin niistä saadaan yksi yhtenäinen affiiinimuunnosmatriisi. Teemme tämän käyttämällä Transform3D-luokan mul- eli multiply-metodia.

```
kaannosXsuhteen.mul(kaannosYsuhteen);
```

Muodostamme itse TransformGroup-haaran antamalla sille luotu affiinimuunnos matriisi parametrina sekä lisäämällä siihen edellisessä esimerkissä luomamme kolmiulotteinen kuutio.

```
muunnoshaara.setTransform(kaannosXsuhteen);

muunnoshaara.addChild(new ColorCube(0.2));
```

Lisäämme vielä TransformGroup-haaran olemassa olevaan BranchGroup-haaraan.

```
BranchGroup ryhmasolmu = new BranchGroup();
ryhmasolmu.addChild(muunnoshaara);
```

BranchGroup-luokalla on myös metodi compile, joka rakentaa maisegraafista yhtenäisen kokonaisuuden yhdistäen useammat samassa haarassa olevat Transform3D-oliot yhdeksi, jolloin maisemagraafi on hieman yksinkertaisempi ja nopeampi käsitellä. [41]

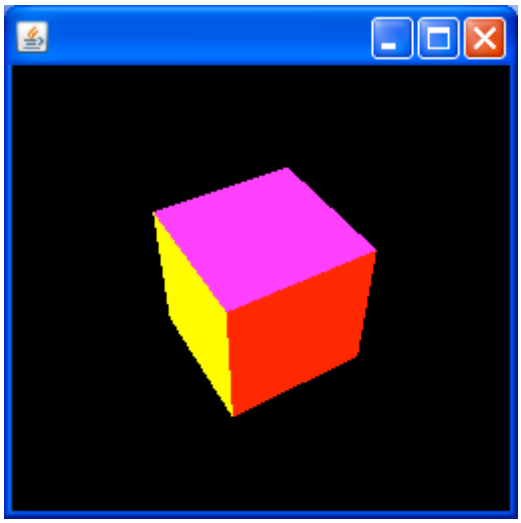
```
ryhmasolmu.compile();
```

Lopuksi liitämme ryhmäsolmun(t) kiinni olemassa olevaan universumiin

```
universumi.addChild(ryhmasolmu);
```

Ja maisemagraafi on valmis piirettäväksi.

Näin olemme saaneet käännettyä kuutiota 45-astetta x-akselin suhteen, sekä 30-astetta y-akselin suhteen, kuva 6.



Kuva 6 Kääntö

Sama käännös voitaisiin suorittaa myös käyttämällä Transform3D-luokan metodia setEuler, joka suorittaa käännöksen Eulerin kulmien (cos alfa + isin alfa)(x+iy) alfa-kulmien perusteella. Metodille setEuler annetaan parametrina Vector3d–olio, jossa käännösakselit ilmaistaan x,y,z-parametreina.

Mikäli muunnoshaaraan lisättäisiin toinen kappale addChild-metodilla, kääntyisi sekin Transform3D-olion määrittelemällä tavalla. Tosin tämä toinen kappale ei näkyisi kuvassa tai peittäisi siinä nyt olevan kuution, sillä niillä on samat koordinaatit. Siksi mahdollinen uusi kolmiulotteinen kappale pitäisi li-

sätä ensin omaan TransformGroup-haaraansa, jossa sillä olisi omat koordinaatit, ja sitten vasta tämä haara tulisi lisätä edelliseen muunnoshaaraan.

Edellisen esimerkin lähdekoodi löytyy viitteestä [10] .

3.2 Siirto

Kuten aiemmin on todettu, Java 3D:ssä määritellään kappaleiden sijainnit avaruudessa käyttämällä x, y, z -koordinaatistoa. Java 3D käyttää siirron laskemiseen matriiseja, joissa lisätään vanhoihin x, y, z-pisteisiin uudet pisteet x, y, z, jolloin saadaan uudet siirretyt koordinaatit x', y', z'.

Toteutamme siirron vektorin (0.5, 0.5, 0) suhteen kertomalla äskeisen Transform3D matriisin toisella Transform3D-matriisilla, joka sisältää uudet koordinaatit.

Aloitamme luomalla siirtovektorin käyttämällä Vector3f-luokkaa (Annamme arvot liukulukuina, mutta käyttää voisi myös kokonaislukuja Vector3d-luokan yhteydessä).

```
Vector3f siirtoVektori = new Vector3f(-0.5f,0.5f,0);
```

Ote Vector3f-luokan dokumentaatiosta:

> "A 3-element vector that is represented by single-precision floating point x ,y ,z coordinates." [11]

Sitten luomme olion siirto, jolle määrittelemme siirtovektorin parametriksi.

```
Transform3D siirto = new Transform3D();
siirto.setTranslation(siirtoVektori);
```

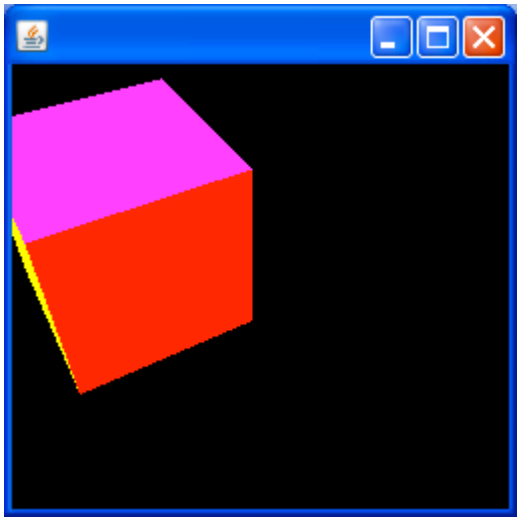
Lopuksi kerromme edellisessä esimerkissä tehdyn matriisin siirtolausekkeen sisältävällä matriisilla.

```
kaannosXsuhteen.mul(siirto);
```

Tämä sisälsi myös edellisen käännöksen akselin-Y suhteen.

Kun tämä sitten lisätään muunnoshaaraan, joka sisältää kuution, saadaan siirrettyä kappaletta koordinaatistossa vektorin (-0.5, 0.5, 0) suhteen.

```
muunnoshaara.setTransform(kaannosXsuhteen);
ryhmasolmu.addChild(muunnoshaara);
universumi.addChild(ryhmasolmu);
```



Kuva 7 Siirto

Mikäli lisäisimme kuvaan muita kappaleita, täytyisi niille antaa omat muunnoshaarat, koska muutoin niiden yläpuolella olevat TransformGroup-oliot vaikuttaisivat myös niihin.

Lisäämme esimerkkiimme HelloWorld [10] toisen kuution, jolla on jo siirto vektorin (-0.2, 0.2, 0) suhteen.

Luomme vektorin sekä tarvittavat haarat:

```
Vector3f siirtoVektori2 = new Vector3f(-0.2f,0.2f,0);
TransformGroup alaHaara = new TransformGroup();
Transform3D siirto2 = new Transform3D();
```

Asetamme muunnokselle siirron vektorin(-0.2,0.2,0) suhteen, sekä asetamme sen alahaaralle. Lisäksi lisäämme alahaaraan kuution.

```
siirto2.setTranslation(siirtoVektori2);
alaHaara.setTransform(siirto2);
alaHaara.addChild(new ColorCube(0.2));
```

Kun lisäämme tämän alahaaran aiempaan muunnoshaaraan, käytetään addChild metodia:

```
muunnoshaara.addChild(alaHaara);
```

Tällöin siitä tulee muunnoshaaran lapsisolmu ja alahaarasolmussa oleva kuutio siirtyy tällöin ensin vektorin (-0.5,0.5,0) suhteen ja sitten vektorin (-0.2, 0.2,0) suhteen eli sen uudet koordinaatit ovat pisteissä (-0.7,0.7,0). Täydellinen lähdekoodi löytyy viitteestä 12.

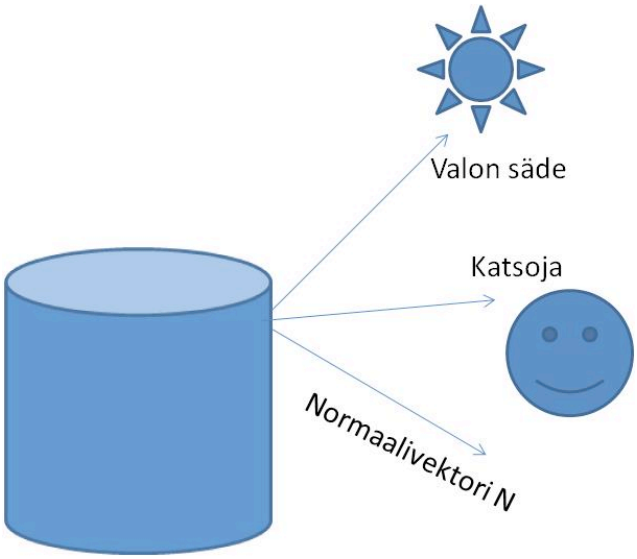
4 VALAISTUSMALLIT JA SÄVYTYS

Kappaleiden kolmiulotteiseen vaikutelmaan vaikuttavat katselukulman lisäksi myös valaistusolosuhteet. Kun kolmiulotteiseen malliin lisätään valonlähde, joudutaan tällöin huomioimaan jokaisen valonsäteen vaikutus ympäristöön. On huomioitava, kuinka kappaleet ja niiden pintamateriaalit käsittelevät niihin osuvaa valonsädettä, absorboivatko ne valoa vai heijastavatko ne kenties sitä takaisin tai mahdollisesti jopa ohjaavat valonsäteen itsensä kautta toiseen kappaleeseen, joka puolestaan käsittelee siihen osuvaa valonsädettä uudelleen. Lisäksi täytyy muistaa huomioida valonsäteen saapumiskulma. Tämä on laskennallisesti erittäin vaativaa, mutta yksinkertaistamalla mallia sekä käyttämällä hyväksi katseluikkunaa (spatial boundary) saadaan laskennallinen vaativuus pienenemään huomattavasti. Usein kolmiulotteisessa reaaliaikaisessa piirtämisessä käytettävä valaistusmalli on ns. paikallinen valaistus, jossa huomioidaan ainoastaan pisteeseen suoraan tuleva valo, jolloin ympäristössä olevat muut valoa heijastavat kappaleet eivät vaikuta valaistusmalliin lainkaan.

Java 3D käyttää valaistusmallia, jossa lasketaan pinnan normaalivektori, valon suunta sekä kappaleen sijainti katsojan suhteen. Java 3D:n käyttämä valaistusmalli ei huomioi kappaleiden toisiinsa heijastamaa valoa, sillä laskentatehon säästämiseksi valaistusmalli laskee jokaisen kappaleen valaistuksen yksitellen. Valon lähteet käsitetään pistemäisiksi, jolloin ne heijastavat vain suoraa valoa, joka on yhtä suuri kaikkialla.

Tarkempi matemaattinen kuvaus Java 3D:n käyttämästä valaistusmallista löytyy Java 3D -rajapinnan API-dokumentaatiosta viitteessä 13.

Kuvassa 8 on kuvattuna Java 3D -valaistusmalli.



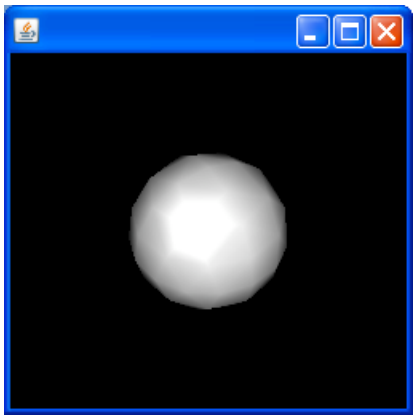
Kuva 8 Java 3D valaistusmalli

Sävytyksellä tarkoitetaan prosessia, jossa lasketaan yhden pikselin tarkka väriarvo, minkä jälkeen muiden pikseleiden väriarvot saadaan interpoloimalla viereisten pikseleiden väriarvoista uudet. Java 3D:n valomallissa jokaisen pikselin värisävy lasketaan kyseiseen pikseliin osuvien valonsäteiden summan avulla. Väriarvojen määrityksessä Java 3D mallintaa värin laskemalla yhteen valon värin sekä materiaalin värin yhdistelemällä punaista, vihreää ja sinistä. Tämä ei ehkä ole tarkka tapa määritellä väriä, eikä se sisällä ihan kaikkia värisävyjä, mutta se on riittävä luomaan optisen illuusion aidoista väreistä tietokonelaitteiston puutteiden rajoissa.

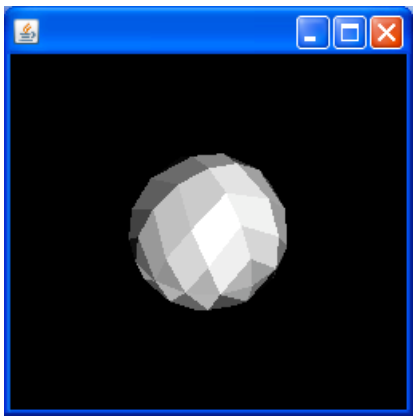
Sävytykseen Java 3D tarjoaa kahta eri sävytysmallia, SHADE_GOURAUD, SHADE_FLAT. Gouraud-sävytys näyttää hienoimmalta, koska jokainen pikseli sävytetään käyttäen trilineaarista interpolaatioita[40] viereisen pikselin sävytysarvosta. Flat-sävytysmalli taas on karumman näköinen, mutta erittäin nopea, varsinkin ohjelmistopohjaisessa piirtämisessä, jossa ei voida käyttää hyväksi näytönohjaimen 3D-piirto-ominaisuuksia. Flat-sävytyksessä monikulmion pikseleiden väriarvot lasketaan suoraan monikulmion normaalivektorin kulman sekä monikulmioon osuvan valonsäteen kulman avulla. Näiden lisäksi tarjolla on myös Phong-sävytys, jossa monikulmion jokaiselle kärkipisteelle lasketaan oma pintaa kohtisuorassa oleva normaalivektori, joista sitten interpoloimalla saadaan laskettua väriarvot sekä kirkkaudet monikulmion jokaiselle pikselille. Tämä on kuitenkin laskennallisesti vaativaa ja sen käyttämiseen tarvitaan vielä tällä hetkellä näytönohjainvalmistajalta saatava

erillinen CG toolkit. Joten Phong-sävytyksen käytön jätän käsittelemättä tässä työssä.

Sävytysten erot ilmenevät alla olevista kuvista, joista ylempi kuvaa SHADE_GOURAUD ja alempi SHADE_FLAT sävytysmallia.



Kuva 9 SHADE_GOURAUD



Kuva 10 SHADE_FLAT

4.1 Valoitusmalliesimerkki

Seuraavana on yksikertainen valoitusmalliesimerkki, jossa pallon muotoiseen kappaleeseen heijastetaan suora valonsäde. Käytämme Java 3D:n tarjoamaa pallo-oliota. Lisäksi tarvitaan Apperance-olio ilmoittamaan kappaleella olevia ominaisuuksia kuten väriä tekstuuria, materiaalia tai renderöintiparametreja.

Ote Appearance-luokan dokumentaatiosta:

> ”The Appearance object defines all rendering states that can be set as a component object of a Shape3D node.” [14]

Käytämme valonheijastuksen demonstroimiseen yksinkertaista palloa, joka saadaan kätevästi luotua Sphere-luokasta. Luomme myös instanssin Appearance-luokasta.

```
Sphere pallukka = new Sphere(0.4f);
Appearance nakyma = new Appearance();
```

Lisäksi kappaleen näkymään täytyy lisätä parametrina materiaali-olio, jotta valaistusmallin seurauksena ei tule pelkkää valkoista täplää.

```
Material materiaali = new Material();
nakyma.setMaterial(materiaalii);
pallukka.setAppearance(nakyma);
```

Ote Material-luokan dokumentaatiosta:

> ”The Material object defines the appearance of an object under illumination. If the Material object in an Appearance object is null, lighting is disabled for all nodes that use that Appearance object.” [15]

Kun olemme määritelleet kappaleen ja sen materiaalin, lisäämme itse valaistusmallin käyttämällä DirectionalLight-luokkaa, joka esittää kaukaisuudessa olevaa kohdistettua valon lähdettä. Valaistusmallioliolle täytyy muistaa myös asettaa näkymäikkuna (BoundingSphere), jotta Java 3D tietää asettaa sen piirrettäväksi.

```
DirectionalLight valaistusmalli = new DirectionalLight();
valaistusmalli.setInfluencingBounds(new BoundingSphere());
```

Ote DirectionalLight--luokan dokumentaatiosta:

> ”A DirectionalLight node defines an oriented light with an origin at infinity. It has the same attributes as a Light node, with the addition of a directional vector to specify the direction in which the light shines. A directional light has parallel light rays that travel in one direction along the specified vector. Di-rectional light contributes to diffuse and specular reflections, which in turn depend on the orientation of an object's surface but not its position. A direc-tional light does not contribute to ambient re-flections.” [16]

Lopputulos näyttää samalta kuin kuvassa 9.

GOURAUD-sävytys on vakiosävytysmalli, ellei sitä erikseen vaihdeta kutsumalla Appearance-luokan setColouringAttributes-metodia. Kutsuttaessa setColoringAttributes-metodia sille pitää antaa parametrina ColouringAttributes-olio, jossa on asetettuna setShadeModel-parametri. Viitteestä 17 löytyy tämän kappaleen täydellinen esimerkki ohjelmakoodi.

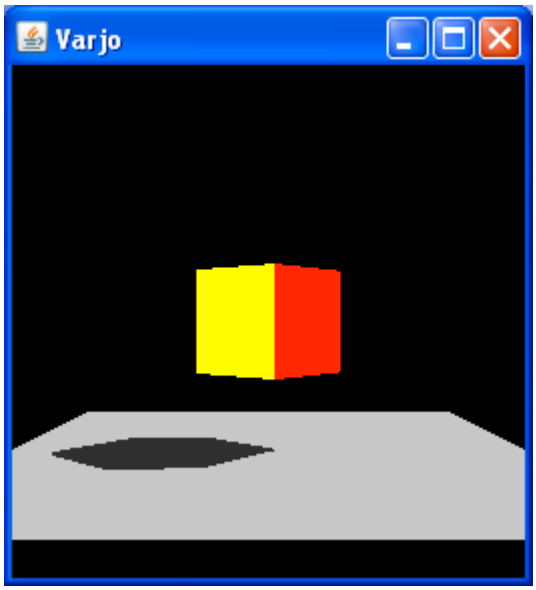
5 VARJOT

Kappaleiden langettamat varjot ovat kuvan luonnollisuuden kannalta merkittävä elementti mutta niiden sisältäminen kuvaan on laskennallisesti varsin vaativa ominaisuus. Kolmiulotteisessa tietokonegrafiikassa varjoja voidaan simuloida laskemalla varjojen suurpiirteinen sijainti ja asettamalla varjoa esittävä tekstuuri niille kohdin, missä varjon tulisi olla. Varjo siis ei ole mikään oikea varjo, vaan pikemminkin tekstuuri tai kaksiulotteinen kappale, jonka tehtävänä on varjon esittäminen.

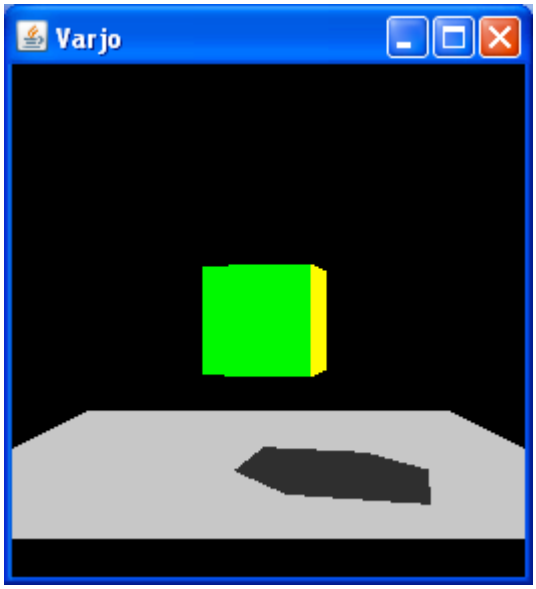
Varjo voidaan lisätä asettamalla varjotekstuuri halutuille monikulmioille. Tällöin tulee huomata, että kun varjon alainen kappale kiertyy tai siirtyy, täytyy myös siihen liitetyn varjon siirtyä mukana. Tämä lisää laskennan määrää ja voi helposti hidastaa piirtämistä, mikäli varjoja on piirrettävässä kuvassa paljon. Kaksiulotteisen varjo-kappaleen ollessa kyseessä voi myös helposti muodostua ongelmatilanne, jossa ”varjo” piirtyykin väärin tai jopa peittää läpinäkymättömyydellään jonkin toisen kappaleen rikkoen täten tehdyn illuusion.

Kuvissa 11-12, on kaksi kuvaa esimerkkikoodista, joka piirtää pyörivän kuution ja sitä seuraavan varjon. Huomataan varjon automaattinen muutos kappaleen katselukulman suhteen. Vaikka kuvassa näyttääkin siltä, että varjo

olisi luotu käyttämällä pyörivää valon lähdettä, on valon lähde oikeasti koko-ajan samassa paikassa. Itse varjo on ColorCube-olion lailla Shape3D-luokasta periytyvä oma olio, joka on lisätty samaan TransformGroup-haaraan, RotationInterpolator-olion sekä ColorCube-olion kanssa. Näin ollen saadaan aikaiseksi kömpelö efekti, että valon lähde pyörisi ja varjo sen mukaisesti, mutta todellisuudessa varjo on vain kaksiulotteinen kappale, joka seuraa kuutiota.



Kuva 11 Varjo efekti



Kuva 12 Varjo efekti

Täydellinen lähdekoodi löytyy liitteestä 18.

6 ANIMOINTI

Java 3D:ssä on Behavior-luokka, joka tarjoaa keinot animointiin sekä erilaisiin herätteisiin reagoimiseen. Kun käyttäjä on kerran määritellyt Behavior-olion, osaa järjestelmä tämän jälkeen automaattisesti päivittää haluttuja attribuutteja kuten sijaintia, orientaatiota, kokoa, väriä tai läpinäkyvyyttä.

Maisemagraafiin voi sijoittaa myös useita erilaisia behavior–olioita. Näiden haittana tosin on, että ne syövät tällöin paljon prosessori-aikaa, joka taas puolestaan hidastaa ruudun päivitystä. Tämän takia kolmiulotteisessa piirtämisessä käytetään ns. katseluikkunaa (spatial boundary), eli ohjelmoija määrittelee katseluikkunan, jonka sisällä olevat asiat tapahtuvat, muutoin niitä ei suoriteta ennen kuin ne osuvat katseluikkunaan. Esimerkiksi metsässä kaatuva puu ei kaadu koskaan, ellei sitä ole joku näkemässä. Katseluikkuna asetetaan käyttämällä Behavior-luokan setSchedulingBounds-metodia, jolle annetaan parametrina joko BoundinSphere-, BoudingBox- tai BoundingPolytype-luokan olio.

6.1 Automaattinen kappaleen pyörittäminen RotationInterpolator-luokalla

Alla olevassa esimerkissä asetetaan kuutio pyörimään katsojan näkökulmasta x-akselin suhteen ympäri. Toteutamme tämän asettamalla ensin TransformGroup-oliolle kirjoitusoikeudet muokkauksen suhteen päälle, kutsumalla sen setCapability-metodia.

```
TGmuokk.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
```

Seuraavaksi luomme aikariippuvaisen funktion alfa, joka generoi lineaarisessa järjestyksessä loputtomiin(-1) arvoja 0 ja 4000:n välillä. Käytämme tätä oliota nopeus parametrina Behavior-olio RotationInterpolatorille. Sellainen on esimerkiksi luokka Alpha.

Ote Alpha-luokan dokumentaatiosta:

> ”The alpha NodeComponent object provides common methods for converting a time value into an alpha value (a value in the range 0 to 1). The Alpha object is effectively a function of time that generates alpha values in the range [0,1] when sampled:

> f(t) = [0,1]. A primary use of the Alpha object is to provide alpha values for Interpolator behaviors” [19]

Se saadaan käyttöön luomalla siitä uusi instanssi ja määrittelemällä se ikuiseksi loopiksi -1 ja asettamalla yhden kierroksen kestoksi 4000 aikayksikköä.

```
Alpha alfa = new Alpha(-1, 4000);
```

Seuraavaksi luodaan itse operaation suorittava Behavior-olio. Sellainen on RotationInterpolator-luokka, joka automaattisesti kääntää kohdettaan interpoloimalla lineaarisesti uuden kulma-arvon sille annetusta arvosta.

Ote RotationInterpolator-luokan dokumentaatiosta:

> ”Rotation interpolator behavior. This class defines a behavior that modifies the rotational component of its target TransformGroup by linearly interpolating between a pair of specified angles (using the value generated by the specified Alpha object). The interpolated angle is used to generate a rotation transform about the local Y-axis of this interpolator.” [20]

Alustamme sen konstruktorin pelkällä Alpha-luokan oliolla, sekä kohteena olevalla muunnoshaara-oliolla.

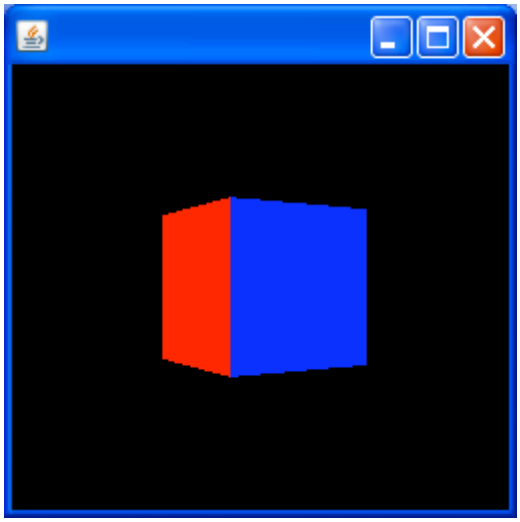
Konstruktorille voisi antaa vielä kohde-akselin määrittelyn Transform3D--luokan setAxisAngle-metodin avulla, sekä minimi ja maksimi-kulmat. Ilman näitä se käyttää oletuksena Y-akselia ja kulmana PI/2-radiaaneja.

```
RotationInterpolator pyorittaja =
    new RotationInterpolator(alfa, TGmuokkaus);
```

Lopuksi määrittelemme katseluikkunan sekä lisäämme Behavior-olion TransformGroup-olion lehdeksi.

```
BoundingSphere akkuna = new BoundingSphere();
pyorittaja.setSchedulingBounds(akkuna);
TGmuokkaus.addChild(pyorittaja);
```

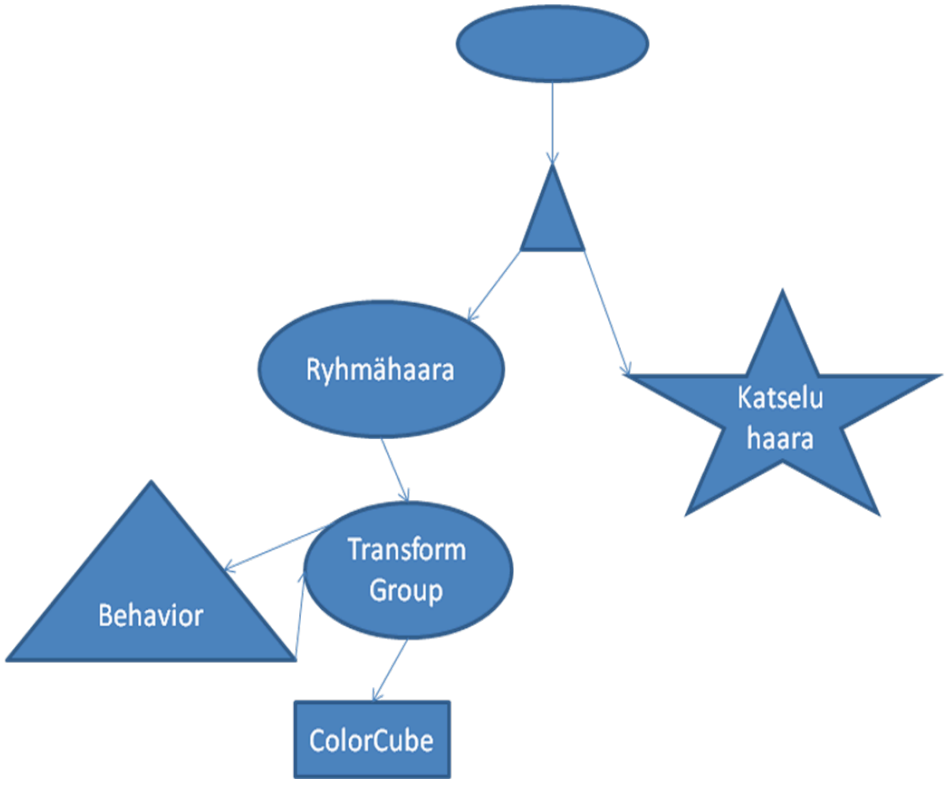
Näin olemme saaneet aikaan x-akselin ympäripyörivän kuution. Kuvassa 13 näkyy yksi frame pyörivästä kuutiosta.



Kuva 13 Rotaatio käyttäen RotationInterpolatoria

Täydellinen lähdekoodi löytyy liitteestä 21

Alla olevassa kuvassa 14 näkyy itse maisemagraafi puumaisessa muodossa.



Kuva 14 Behavior-luokka maisemagraafissa

6.2 Automaattinen siirto käyttäen PositionInterpolator-luokkaa

Java 3D tarjoaa myös muita valmiita toiminto-luokkia animointiin. Esimerkiksi PositionInterpolator-luokka on behavior-luokka, joka laskee kappaleelle sijainnit käyttämällä lineaarista interpolointia sille annetuista pisteväleistä.

Ote PositionInterpolator–luokan dokumentaatiosta:

> ”Position interpolator behavior. This class defines a behavior that modifies the translational component of its target TransformGroup by linearly interpolating between a pair of specified positions (using the value generated by the specified Alpha object). The interpolated position is used to generate a translation transform along the local X-axis of this interpolator.” [22]

PositionInterpolointi-luokkaa kutsutaan yksinkertaisesti luomalla siitä uusi instanssi ja antamalle sille sijaintipisteet, kohdemuunnosryhmä, akselin sisältävä muunnos-olio, jolle se generoi muunnokset, sekä aloitus- ja lopetuskoordinaatit.

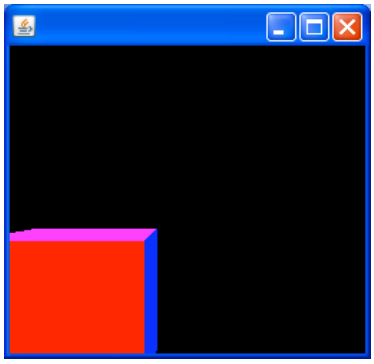
```
PositionInterpolator liikuttaja =
    new PositionInterpolator(alfa, tg, t3d, -1.0f, 1.0f);
```

Muunnos-oliolle täytyy antaa instanssi AxisAngle-luokasta, jonka konstruktorin (double x, double y, double z, double angle) avulla se muodostaa uuden kulman siinä määritellyn vektorin x, y, z ja radiaaneissa olevan kulman angle avulla.

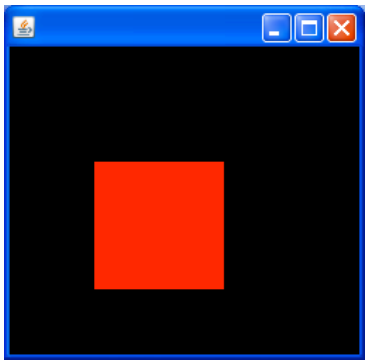
```
Transform3D t3d = new Transform3D();
t3d.set(new AxisAngle4d(5.0,5.0,0.0,Math.PI/2));
```

Mikäli PositionInterpolator-luokkaa käytettäisiin ilman AxisAngle-oliota sekä aloitus ja lopetus koordinaatteja, se liikuttaisi kappaletta Y-akselia pitkin oikealta -1 vasemmalle 1.

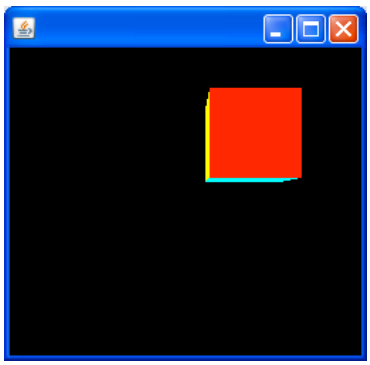
Esimerkiksi yllä oleva akselikulma aiheuttaa sen, että kuutio näyttää siirtyvän katsojasta pois päin.



Kuva 15 PositionInterpolator behavior, kuva 1



Kuva 16 PositionInterpolator, kuva 2



Kuva 17 PositionInterpolator, kuva 3

Katso täydellinen lähdekoodiesimerkki liitteestä 23 PositionInterpolator-luokan käytöstä.

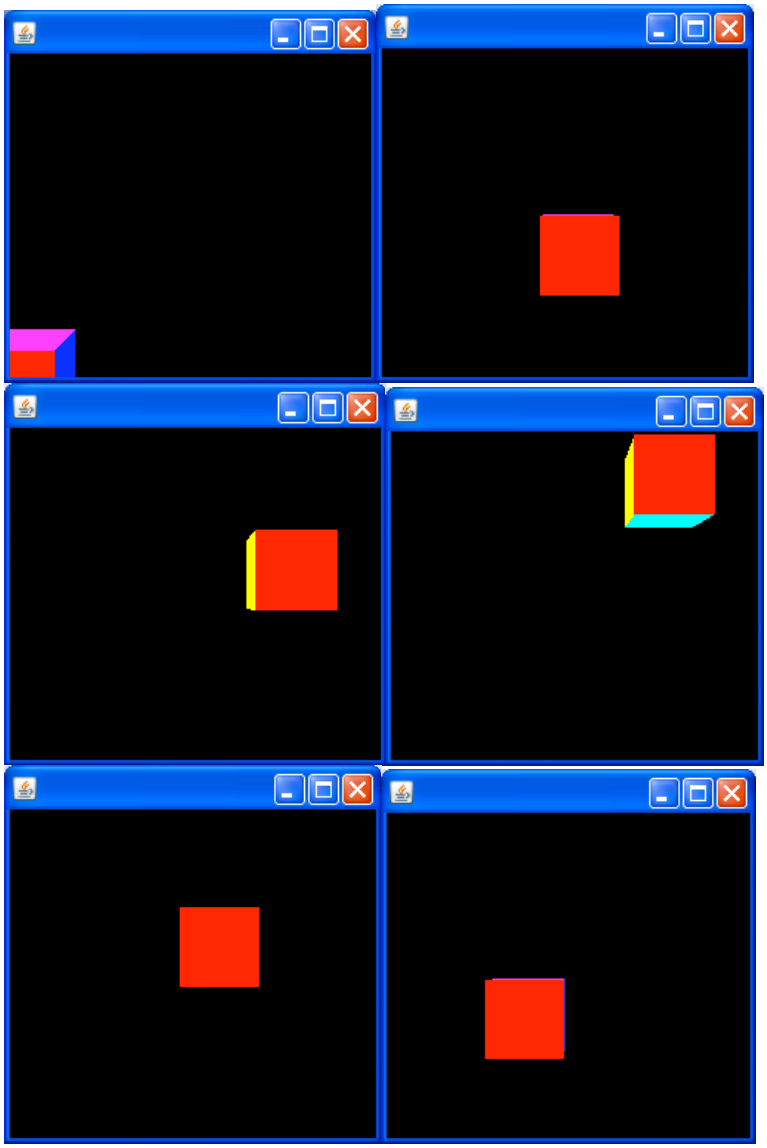
6.3 Automaattinen polunmuodostaminen PositionPathInterpolator-luokalla

PositionInterpolator-luokasta on myös olemassa versio PositionPathInterpolator-luokka, jolle voi määritellä erikseen koordinaatiston pisteet, joita pitkin se suorittaa interpoloinnin käyttäen Alpha-luokan generoimaa arvosarjaa.

Ote PositionPathInterpolator-luokan dokumentaatiosta:

> ”PositionPathInterpolator behavior. This class defines a behavior that mod-ifies the translational component of its target TransformGroup by linearly in-terpolating among a series of predefined knot/position pairs (using the value generated by the specified Alpha object). The interpolated position is used to generate a translation transform in the local coordinate system of this inter-polator. The first knot must have a value of 0.0. The last knot must have a value of 1.0. An intermediate knot with index k must have a value strictly greater than any knot with index less than k” [24]

Seuraavana on kuvasarja PositionPathInterpolator–luokan toiminnasta. Huomataan, että kuutio siirtyy ensin oikeaan ylänurkkaan ja sitten takaisin samaa reittiä vasempaan alanurkkaan.



Täydellinen lähdekoodiesimerkki PositionPathInterpolator-luokan käytöstä löytyy viitteestä [25].

6.4 Automaattinen kierto-sijainti-polku käyttäen RotationPositionPathInterpolator-luokkaa

Lisäksi on olemassa RotPosPathInterpolator Behavior-luokka, jolla on mahdollista toteuttaa sekä kierto että siirto animoituna. PathInterpolator-luokan erikoisuutena on, että se osaa tallettaa arvoja interpolaation laskemista varten. Se voi tallettaa esimerkiksi pisteiden arvoja, joita sitten käytetään hyväksi interpoloinnissa, jolloin pisteen arvo sekä alfan generoimat parametrit määrittelevät animaation kulun.

Ote RotPosPathInterpolator-luokan dokumentaatiosta:

> ”RotPosPathInterpolator behavior. This class defines a behavior that mod-ifies the rotational and translational components of its target TransformGroup by linearly interpolating among a series of predefined knot/positon and knot/orientation pairs (using the value generated by the specified Alpha object). The interpolated position and orientation are used to generate a transform in the local coordinate system of this interpolator.” [26]

Seuraavassa ohjelmanpätkässä animoimme ColorCube-olion liikkumaan ja kääntyilemään pisteiden koordinaateissa. Ensin luomme ryhmä-haaran (objRoot) sekä muunnos-haaran (target). Lisäksi määrittelemme alfan parametreiksi -1, -4000.

```
BranchGroup objRoot = new BranchGroup();
TransformGroup target = new TransformGroup();
```

TransformGroup-haaralle täytyy asettaa kirjoitusoikeudet päälle.

```
tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
Alpha alpha = new Alpha(-1, 40000);
Transform3D t3d = new Transform3D();
t3d.set(new AxisAngle4d(0,0.0,0.0,0.0));
```

Talletamme solmujen arvot liukuluku taulukkoon, käännöspisteet Quat4f-taulukkoon sekä sijainnit Point3f-taulukkoon.

```
float[] solmut = {0.0f, 0.2f, 0.3f, 0.4f, 0.6f, 0.7f, 0.8f,
0.9f, 1.0f};

Point3f[] sijainnit = new Point3f[9];
sijainnit[0]= new Point3f(  0.0f,  1.0f,  0.0f);
sijainnit[1]= new Point3f(  0.5f,  0.5f,  0.0f);
sijainnit[2]= new Point3f(  0.0f,  0.0f,  0.0f);
sijainnit[3]= new Point3f(  -0.5f, -0.5f,  0.0f);
sijainnit[4]= new Point3f(  -1.0f, -1.0f,  0.0f);
sijainnit[5]= new Point3f(  -0.5f, -0.5f,  0.0f);
sijainnit[6]= new Point3f(  0.0f, -0.0f,  0.0f);
sijainnit[7]= new Point3f(  0.5f,  0.5f,  0.0f);
sijainnit[8]= new Point3f(  0.5f,  1.0f,  0.0f);

Quat4f[] quats = new Quat4f[9];
quats[0] = new Quat4f(0.0f, 1.0f, 1.0f, 0.0f);
quats[1] = new Quat4f(1.0f, 0.0f, 0.0f, 0.0f);
quats[2] = new Quat4f(0.0f, 1.0f, 0.0f, 0.0f);
quats[3] = new Quat4f(0.0f, 1.0f, 1.0f, 0.0f);
quats[4] = new Quat4f(1.0f, 0.0f, 0.0f, 0.0f);
quats[5] = new Quat4f(0.0f, 1.0f, 0.0f, 0.0f):
quats[6] = new Quat4f(0.0f, 1.0f, 1.0f, 0.0f);
quats[7] = new Quat4f(1.0f, 0.0f, 0.0f, 0.0f);
quats[8] = new Quat4f(0.0f, 1.0f, 1.0f, 0.0f);
```

Annamme nämä parametrit sekä Alpha-olion, kohteena olevan TransformGroup-olion sekä Transform3D-olion RotPosPathInterpolator-luokan konstruktorille.

```
RotPosPathInterpolator liikuttaja = new RotPosPathInterpolator
    (alfa, tg, t3d, solmut, quats, sijainnit);
```

Lopuksi asetamme sille katseluikkunan käyttäen BoundingSphere-luokkaa, lisäämme solmut puuhun sekä luomme kohteena olevaan TransformGroup haaraan vielä instanssin ColorCube-oliosta.

```
BoundingSphere akkuna = new BoundingSphere();
liikuttaja.setSchedulingBounds(akkuna);
```

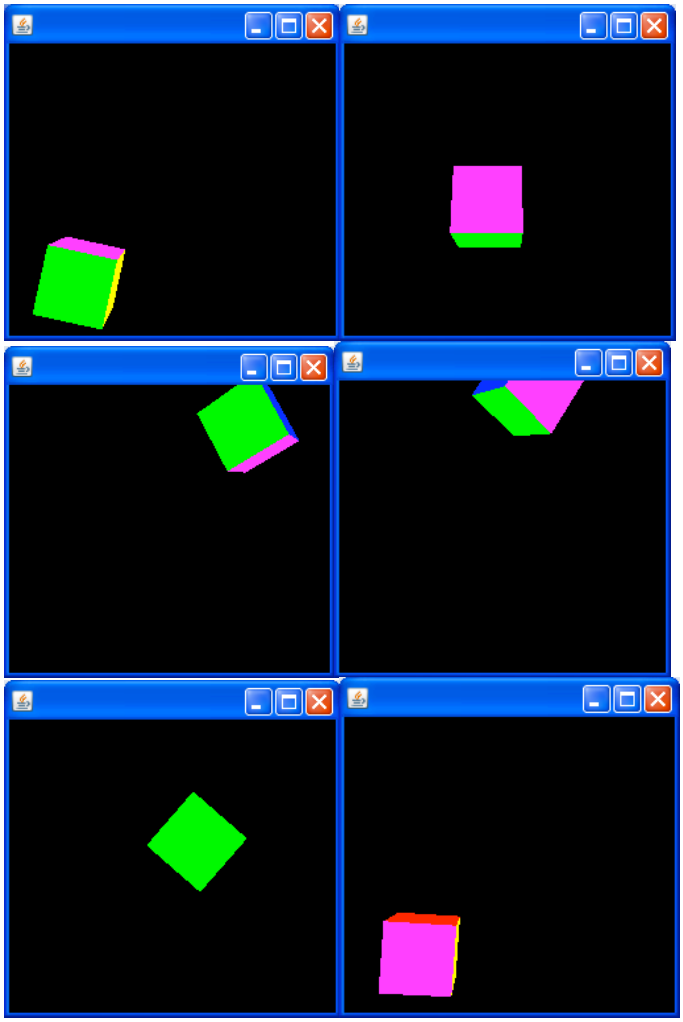
Muodostetaan muunnoshaara lisäämällä siihen solmut kuutio ja liikuttaja.

```
ColorCube kuutio = new ColorCube(0.2);
tg.addChild(kuutio);
tg.addChild(liikuttaja);
```

Lopulta liitämme muunnoshaaran ryhmähaara solmuun, kutsumme sen compile metodia haarojen ja solmujen optimoimiseksi, sekä aktivoimme vielä juurihaaran katselukulman ja lisäämme ryhmähaarasolmun juureen kiinni.

```
group.addChild(tg);
group.compile();
universe.getViewingPlatform().setNominalViewingTransform();
universe.addBranchGraph(group);
```

Kun lopullinen koodi ajetaan, näyttää se alla olevalta kuvasarjalta. ColorCube-kappale liikuu pisteiden mukaisesti, kääntäen aina pisteelle tultaessa akseliaan katsojaa kohdin.



Täydellinen koodiesimerkki löytyy viitteestä [27].

7 INTERAKTIO

Interaktiolla tarkoitetaan tilannetta, jossa käyttäjältä tai muusta lähteestä tuleva syöte aiheuttaa muutoksen virtuaalimaailmassa. Java 3D:ssä toimintoluokka (Behavior) vastaa interaktiosta kuten myös animoinnista.

Behavior-luokka on luokka, joka toteuttaa abstraktin Behavior-luokan metodit alustus (initialization) sekä heräte (prosessStimulus). Behavior-luokan alustusmetodia kutsutaan, kun Behavior-luokan sisältävä maisemagraafi aktivoituu. Alustusmetodi sisältää herätysehdon (WakeupCondition), jonka tapahtuessa kutsutaan sitten itse heräte-metodia, jonka tehtävänä on muokata maisemagraafia toivotunlaisesti.

Behavior-luokan tulee sisältää

- konstruktori, jonka avulla välitetään viittauks muutettavaan olioon (esimerkiksi TransformGroup).

- heräämisehdon sisältävä public void initialization() -metodin määritys.

- public void processStimulus() -metodin määritys, jonka pitää selvittää herätysehto, toteuttaa muutos ja lopuksi nollata herätysehto.

Esimerkkinä behavior-runko, joka muuttaa näppäinpainalluksen havaittuaan kuution kulmaa katselukulman suhteen.

```
public class ToimintoEsimerkki1 {
    public class ToimintoEsimerkki1 extends Behavior{
        private TransformGroup muunnosRyhma;
        private Transform3D kaannos = new Transform3D();
        private double kulma = 0.0;

        // Konstruktori joka sisältää viittauksen kohteeseen.
        SimpleBehavior(TransformGroup kohde){
            this.kohde = muunnosRyhma;
        }

        // Alustusmetodi
        public void initialize(){
            // Herätysehto
            this.wakeupOn( new
                    WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
        }

        // Itse herätemetodi
```

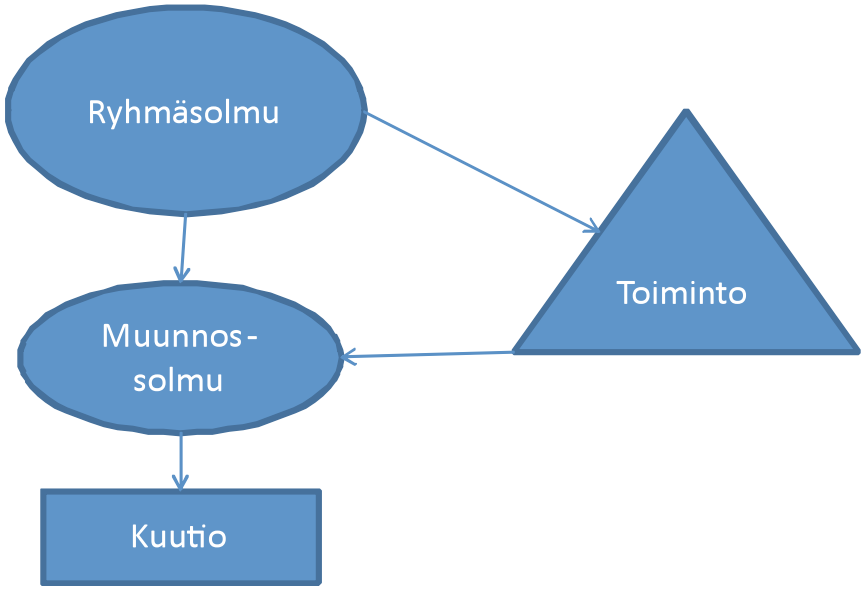
```
        public void processStimulus(Enumeration criteria){
            kulma += Math.PI/6;
            kaannos.rotY(kulma);
            muunnosRyhma.setTransform(kaannos);

        // nollataan herätysehto seuraavaa kertaa varten.
        this.wakeupOn(
                new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
    }
}
```

Lisättäessä Behavior-olio maisemagraafiin, täytyy sille määritellä tapahtumaikkuna kutsumalla setSchedulingBounds-metodia.

Lopuksi olisi hyvä myös kutsua itse universumin toteuttavalle haaralle compile-metodia, jolloin Java 3D optimoi maisemagraafin suoritusta varten.

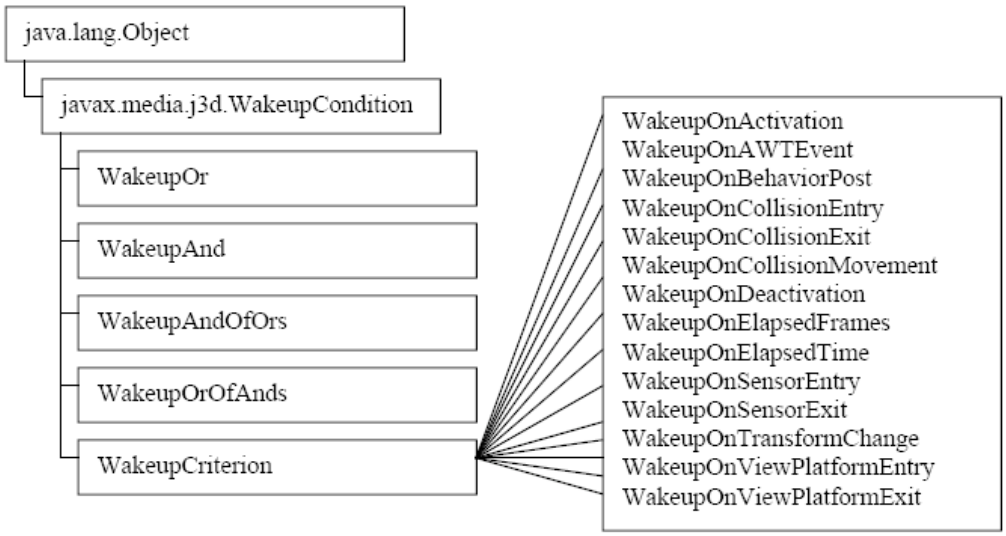
Alla on kuvattuna maisemagraafi, johon on lisätty Behavior-olio. Kaaviossa ryhmäsolmu kutsuu Behavior-oliota, joka sitten muokkaa muunnossolmua täten aiheuttaen muunnoksen myös muunnossolmun alla olevalle kuutiolle.



Kuva 18 Behavior-olion sisältävä maisemagraafi

Heräte-ehto (WakeupCondition) on luokka, jolla hallitaan herätteiden käsittelyä. Sen abstrakti perusluokka WakeupCondition tarjoaa metodit allElements ja triggeredElements, joiden avulla saa enumeroidun listan herätteistä. Lisäksi on myös WakeupCriterion-luokka, joka tarjoaa metodin hasTriggered, jolla hallitaan herätteen jälkeistä tilannetta.

Alla olevassa kuvassa on lueteltu WakeupCondition-luokasta periytyvät luokat ja niiden hierarkinen kuvaus.



Kuva 19 Heräte-ehto -luokan hierarkiakuvaus (Bouvier, 2001)

7.1 Hiiri

Tuki hiirelle löytyy myös behavior-luokista, erityisesti MouseBehavior-nimisestä luokasta.

Ote MouseBehavior-luokan dokumentaatiosta:

> ”Base class for all mouse manipulators (see MouseRotate, MouseZoom and MouseTranslate for examples of how to extend this base class)”

Tähän luokkaan pohjautuen Java 3D:stä löytyy kolme perushiiritoimintoluokkaa, jotka löytyvät paketista com.sun.j3d.utils.behaviors.mouse:

- MouseRotate rotatoi kappaletta, kun hiiren vasenta painiketta painetaan.

- MouseTranslate siirtää kappaletta, kun hiiren oikeanpainiketta painetaan.

- MouseZoom skaalaa kappaleen kokoa, kun hiiren vasen painike sekä ALT näppäin ovat painettuina.

Hiiritoimintoluokat kuuntelevat Javan AWT -kirjaston hiiritapahtumia, kuten hiiren näppäimen painamista, näppäin pohjassa liikuttamista tai näppäimen vapautusta.

Hiiren kursorin x, y -koordinaatit muutetaan automaattisesti Transform3D-olioksi, joka syötetään TransformGroup-haaraan, jolloin jokainen haaran alla oleva lapsi reagoi hiiren syötteeseen.

Jos haluaisimme esimerkkiimme hiiren vasemman näppäimen painallukses-ta rotatoituvan kuution, voitaisiin se tehdä seuraavalla tavalla.

Tarvitsemme muunnoshaaran, johon toimintoluokan tulee vaikuttaa.

```
TransformGroup subTg = new TransformGroup();
```

Lisäksi haaralla tulee olla kirjoitusoikeudet asetettuna, jotta sitä voidaan muokata Transform3D-olion toimesta.

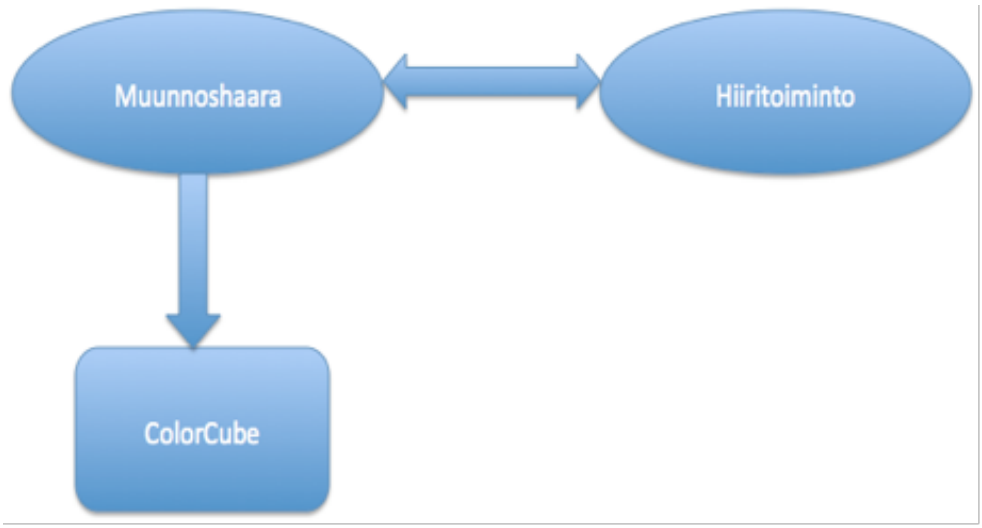
```
subTg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
```

Lisäämme haaraan uuden instanssin ColorCube-luokasta, johon haluamme muunnoksien kohdistuvan.

```
subTg.addChild( new ColorCube(10.0) );
```

Lopuksi lisäämme haaraan MouseRotate Behavior -luokan instanssin, joka suorittaa käännökset aina hiiren vasemman näppäimen ollessa alas painet-tuna. Täydellisen esimerkin lähdekoodin kera löydetään viitteestä 28.

```
MouseRotate mouseRot = new MouseRotate( subTg );
subTg.addChild( mouseRot );
```



Kuva 20 Hiiritoiminto maisemagraafissa

7.2 Näppäinnavigointi

Vastaavasti on myös olemassa valmis KeyNavigatorBehavior-luokka, joka mahdollistaa haaran katselukulman tai kappaleen ohjaamisen näppäimistöllä.

Sen käyttö on varsin helppoa,kun siitä luodaan uusi instanssi,

```
KeyNavigatorBehavior nappainNavigointi =
    new KeyNavigatorBehavior();
```

Asetetaan sille katseluikkuna

```
nappainNavigointi.setSchedulingBounds(katseluikkuna);
```

Liitetään se solmuksi muunnosryhmä tai ryhmähaara-solmuun

```
BranchGrouppi.addChild(nappainNavigointi);
```

Tämän jälkeen katselukulmaa tai kappaletta voidaan liikuttaa nuolinäppäimillä. Täydellisen esimerkin lähdekoodin kera löydetään viitteestä 29.

8 TEKSTUROINTI

Jotta kolmiulotteinen kappale näyttäisi muultakin kuin vain muoviselta yksiväriseltä kappaleelta, täytyy siihen lisätä jokin tekstuuri. Teksturoinnilla tarkoitetaan operaatiota, jossa kappaleeseen lisätään tekstuuri, joka on kaksiulotteinen pikselikuva sekä tekstuurikuvaus, joka määrittää kappaleen jokaisen pisteen x, y, z kaksiulotteiseksi tekstuurikoordinaatiksi u, v.

Näin ollen jokainen pikseli saa tekstuurin siltä kohdin kuin kyseisen alueen tekstuurikuvaukseen on määritelty. Esimerkkinä teksturoinnin tärkeydestä kolmiulotteisessa tietokonegrafiikassa olisi vaikkapa yksivärinen matto siitä huolimatta, että sen jokainen lanka olisi täysin samanvärinen. Ei todellisessa elämässä näytä siltä, kuin se olisi vain yksivärinen kappale johtuen ympäristön valon lähteistä ja valon käyttäytymisestä sen osuessa yksittäiseen lankaan.

Reaaliaikaisissa kolmiulotteisissa mallinnuksessa tällaisen efektin mallinnus olisi niin työläs, ettei siinä ole järkeä nykyisillä laitteistoresursseilla. Siksi efektiä simuloimaan käytetään erilaisia tekstuureita, jotka asetetaan kappaleeseen, jolloin se näyttää hieman todellisemmalta, mutta sen piirtäminen on suhteellisen yksinkertaista, sillä tekstuuri on vain kaksiulotteinen pikselikuva.

Java 3D:ssä tekstuureita käsitellään Appearance-luokan avulla, jolle annetaan parametrina Texture-luokan olio.

Ensin tekstuurikuva täytyy ladata Java 3D -ympäristöön lukemalla se TextureLoader-luokan oliolla, josta se syötetään ImageComponent2D-olioksi.

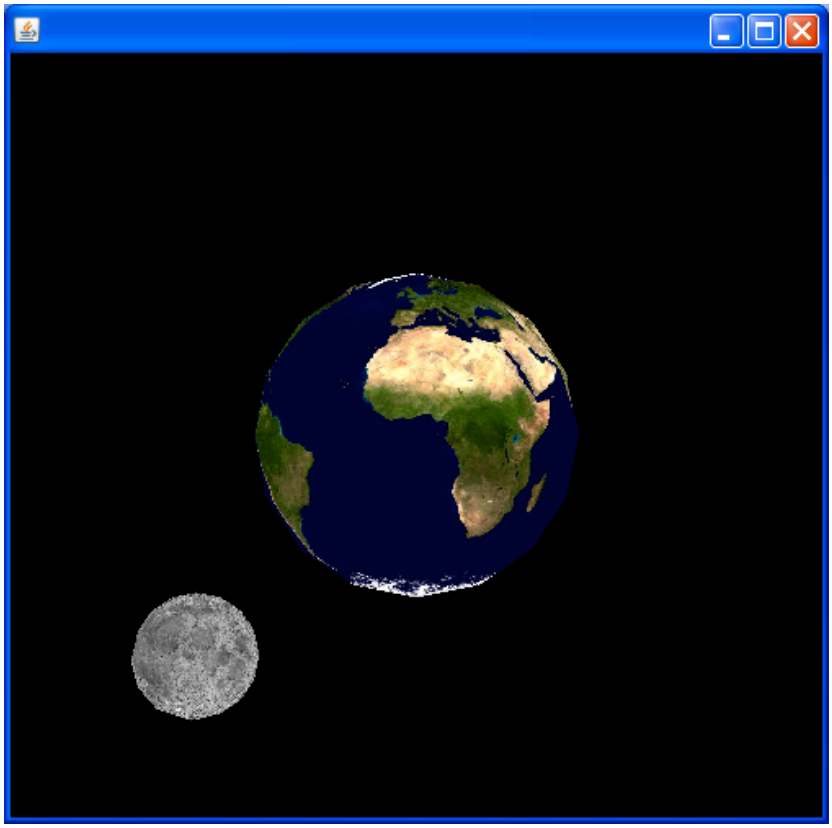
```
TextureLoader lukija = new NewTextureLoader(”tekstuuri.jpg”);
ImageComponent kuva = lukija.getImage();
```

Sen jälkeen siitä voidaan tehdä Texture-luokan olio, joka saadaan puolestaan syötettyä Appearance-luokalle.

```
Texture2D tekstuuri = new Texture2D();
tekstuuri.setImage(0, kuva);
Appearance appear = new Appearance();
appear.setTexture(tekstuuri);
```

Lopulta asetetaan kappaleen Appearance-ominaisuus päälle, yhdessä sen luonnin yhteydessä. Täydellinen lähdekoodi löytyy viitteestä 30.

```
Sphere pallukka =
    new Sphere(0.6f,Primitive.GENERATE_TEXTURE_COORDS,appear);
```



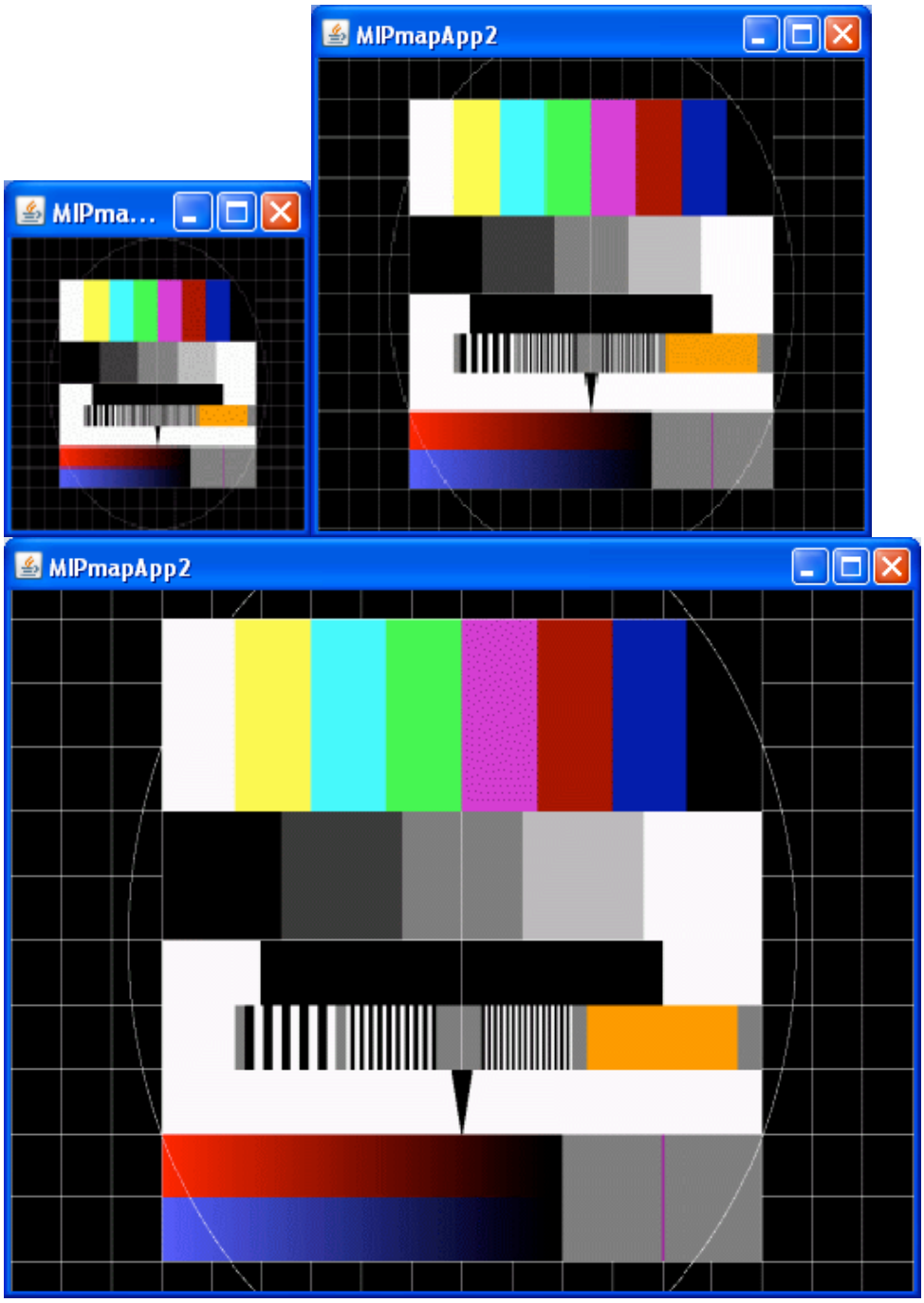
Kuva 21 Teksturoidut pallot

8.1 Mipmap

Mipmap (Multiple levels of texture) on tekniikka, jossa tekstuurista muodostetaan monitasoinen hierarkia, jossa alimpana on alkuperäinen tekstuuri ja jokaisen uuden version sivujen pituus on aina puolet edellisestä. Näin ollen pinon huipulla oleva tekstuuri on vain yhden pikselin kokoinen. Tämä mahdollistaa sen, että monesta pieniresoluutioisesta tekstuurista voidaan kasata yksi isoresoluutioinen tekstuuri.

Tekstuurin mipmäppäystä voisi verrata detaljitason automaattiseen skaalautumiseen, mutta tekstuurina. Näin ollen tekstuuri skaalautuu aina tarpeen mukaan, koska se koostuu pienistä osatekijöistä. Tällä saavutetaan varsin

merkittävää tekstuurimuistin säästöä. Täydellinen esimerkki löytyy viitteestä 31. Kuvassa 22 on demonstroituna tekstuurin mipmäppäys.



Kuva 22 Mipmap-tekniikka

8.2 Mainostaulutekniikka

BillBoard eli mainostaulu tarkoittaa tekniikkaa, jossa staattinen kaksiulotteinen tekstuuri käännetään aina siten, että se on katsojaan päin. Esimerkiksi rallipeleissä olevat metsät koostuvat puista, jotka näyttävät kolmiulotteisilta, mutta lähempää tarkasteltaessa huomataan, että ne kääntävät katsojaa kohdin aina saman tekstuurin, ja ovatkin siten kaksiulotteisia.

Java 3D tarjoaa tätä varten erillisen BillBoard-luokan, jolla voidaan toteuttaa mainostaulumaisia kappaleita, joissa katsoja näkee kappaleen aina samasta kulmasta huolimatta katsojan katselukulman muutoksesta.

Seuraavassa koodiesimerkissä toteutammme kaksiulotteisen puun.

```
BranchGroup objRoot = new BranchGroup();
Vector3f translate = new Vector3f();
```

Määrittelemme kaksi TransformGroup-luokkaa puulle, joista toisen tehtävänä on määritellä puun sijanti ja toisen puun rotaatio.

```
Transform3D T3D = new Transform3D();
TransformGroup TGT = new TransformGroup();
TransformGroup TGR = new TransformGroup();
```

Luomme myös tyhjän BillBoard-tyyppisen olion, määrittelemme katselukulman sekä suuntavektorin puulle.

```
Billboard billboard = null;
BoundingSphere bSphere = new BoundingSphere();
```

Alustamme BillBoard-olion käyttäen rotaation hoitavaa TransformGroup-oliota. Toiseen TransformGroup-olioon (TGT) asetamme sijainnin paikkavektorin (1,1,0) suhteen käyttämällä TransformGroup-luokan set-Translation-metodia sekä lopuksi asetamme katseluikkunan mainostaululle.

```
translate.set(new Point3f(1.0f, 1.0f, 0.0f));
T3D.setTranslation(translate);
TGT.set(T3D);
TGR.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
billboard = new Billboard(TGR);
billboard.setSchedulingBounds(bSphere);
```

Tämän jälkeen kasaamme rakennelman ja lisäämme TGR transformgroup oliolle peruspuun.

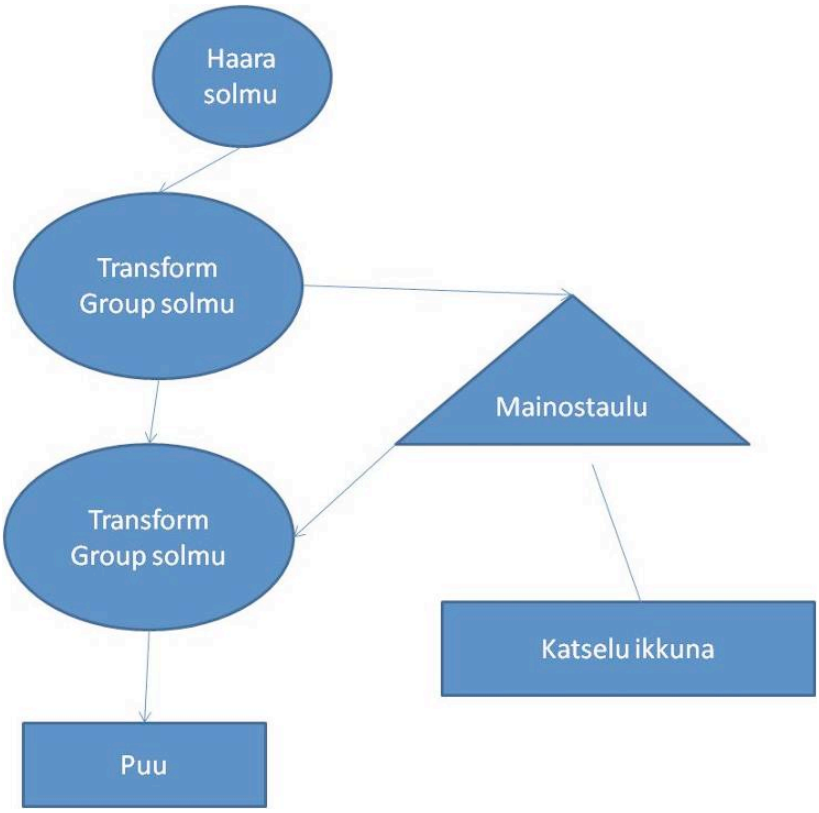
```
objRoot.addChild(TGT);
objRoot.addChild(billboard);
TGT.addChild(TGR);
TGR.addChild(createTree());
```

Alla olevassa kuvassa näkyy kolme ”mainostaulu”-puuta, jotka kaikki näyttävät aina samaa kulmaa katsojalle.

Kuva 23 Mainostaulu esimerkki

Katso viitteestä 32 mainostauluesimerkin täydellinen lähdekoodi.

Kuvassa 24 on kuvattuna mainostauluesimerkin solmujen suhteet maisemagraafi kaaviona.



Kuva 24 Mainostauluesimerkki maisemagraafina

8.3 Detaljitason asteittainen tarkentaminen

LOD (Level of Detail) eli detaljitason asteittainen tarkentaminen on tekniikka, joka mahdollistaa kappaleiden detaljimäärän liukuvan tarkennuksen riippuen kappaleen sijainnista katsojan suhteen. Käytännön esimerkkinä tämä tarkoittaa tilannetta jossa katsoja näkee kaukaa katsoessaan vain metsää, mutta mennessään lähemmäs hän näkee myös puut ja lopulta havunneulaset.

Tekniikka mahdollistaa huomattavan prosessointiajan säästön piirrettävän grafiikan suhteen, jolloin katsojan näkymään voidaan sijoittaa paljon piirrettäviä kappaleita ja saada kuva vielä piirtymään siedettävässä ajassa.

Java 3D käyttää LOD-efektin tuottamiseen LOD-luokkaa sekä Switch-luokkaa, joita voi olla useita per LOD-olio.

DistanceLOD on luokka, joka periytyy LOD-luokasta ja määrittelee animaation tason riippuen kappaleen sijainnista katsojan suhteen. DistanceLOD-luokalle tulee määritellä lista raja-arvoista, joiden kohdilla tehdään muutokset.

Ote DistanceLOD-luokan dokumentaatiosta:

> ”This class defines a distance-based LOD behavior node that operates on a Switch group node to select one of the children of that Switch node based on the distance of this LOD node from the viewer.” [33]

Switch-luokka taas on luokka, jonka tehtävänä on hallita haarassa olevien lapsisolmujen piirtämisvuoroja.

Ote Switch-luokan dokumentaatiosta:

> ”The Switch node controls which of its children will be rendered. It defines a child selection value (a switch value) that can either select a single child, or it can select 0 or more children using a mask to indicate which children are se-lected for rendering. The Switch node contains an ordered list of children, but the index order of the children in the list is only used for selecting the ap-propriate child or children and does not specify rendering order.” [34]

DistanceLOD-luokan käyttö on varsin mutkatonta. Käytämme esimerkkinä Sun Microsystemsin Java 3D -sivuilta löytyvää DistanceLODApp.java -esimerkkiä, jonka täydellinen lähdekoodi löytyy liitteestä.

DistanceLODApp-esimerkissä luodaan Switch–olio, joka määrittelee neljä eri palloa raja-arvoiksi. Lisäksi tulee huomata, että Switch-oliolle täytyy asettaa capability-bitti päälle, jotta sen muokkaus maisemagraafissa onnistuisi.

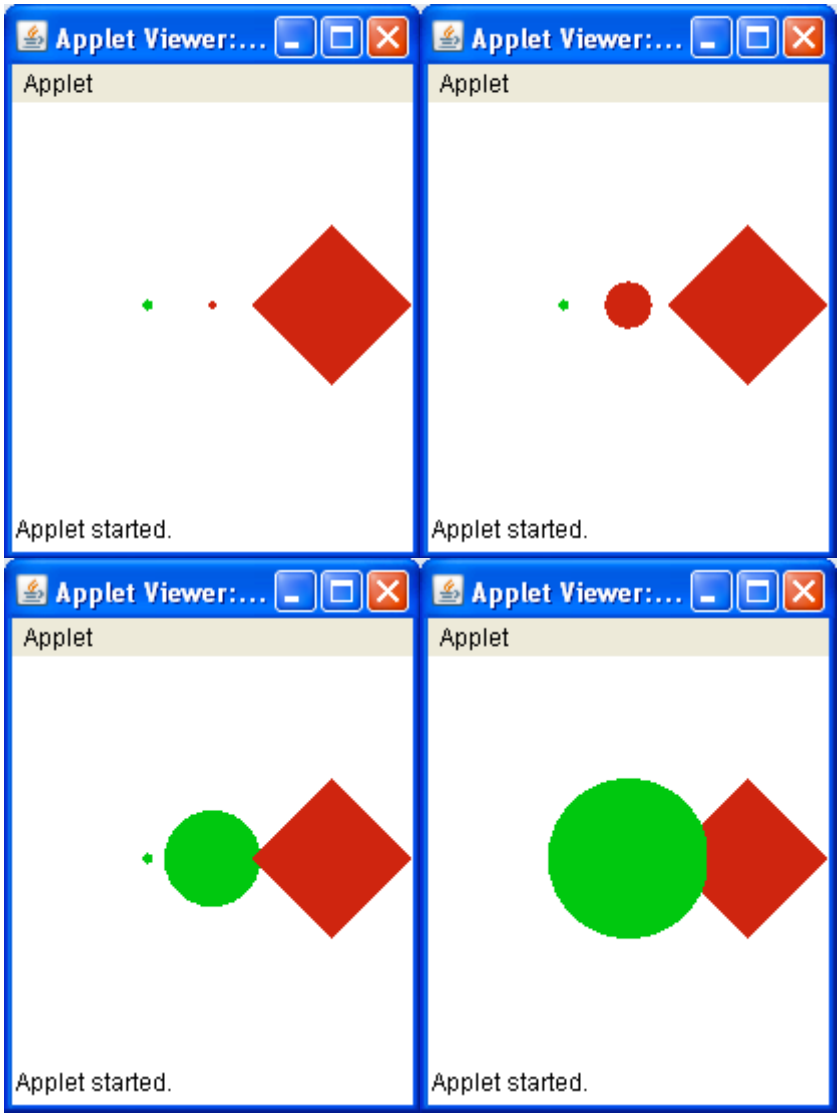
```
Switch targetSwitch = new Switch();
targetSwitch.setCapability(Switch.ALLOW_SWITCH_WRITE);
targetSwitch.addChild(new Sphere(.40f, 0, 25));
targetSwitch.addChild(new Sphere(.40f, 0, 15));
targetSwitch.addChild(new Sphere(.40f, 0, 10));
targetSwitch.addChild(new Sphere(.40f, 0, 4));
```

Seuraavaksi luodaan DistanceLOD–olio, jolle annetaan parametreina etäisyydet sekä Point3f-olio sijantien talletusta varten. Lopuksi vielä lisätään switch-vipu DistanceLOD-oliolle.

```
float[] distances = { 5.0f, 10.0f, 20.0f};
DistanceLOD dLOD = new DistanceLOD(distances, new Point3f());
dLOD.addSwitch(targetSwitch);
```

Lopuksi kasataan maisemagraafi lisäämällä DistanceLOD-luokan instanssi sopivaan ryhmähaaraan BranchGroupiin käyttämällä sen metodia addChild();

Alla olevissa kuvakaappauksissa otteita on Sun Microsystemsin Java 3D -esimerkistä DistanceLODApp.java, joka demonstroi etäisyyden suhteen muuttuvaa ympyräkappaletta.



Kuva 25 Detaljitason asteittainen tarkentaminen

Katso viitteestä 35 esimerkin täydellinen lähdekoodi.

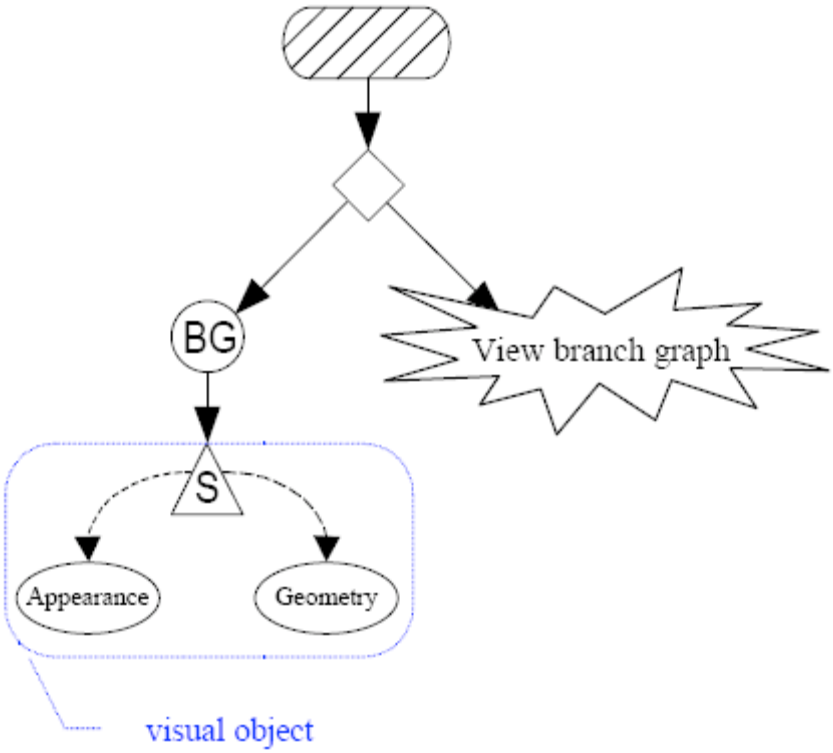
9 OMIEN MALLIEN TUONTI JAVA 3D -MAISEMAGRAAFIIN.

Java 3D:ssä kaikki kolmiulotteiset kappaleet periytyvät Shape3D-luokasta.

Shape3D-luokan instanssi koostuu Geometry- sekä Appearance-luokasta.

Ote Shape3D-luokan dokumentaatiosta:

> "The Shape3D leaf node specifies all geometric objects. It contains a list of one or more Geometry component objects and a single Appearance compo-nent object. The geometry objects define the shape node's geometric data. The appearance object specifies that object's appearance attributes, includ-ing color, material, texture, and so on." [36]



Kuva 26 Shape3D-instanssin rakenne (Bouvier 2001)

Jotta ulkopuolisten kolmiulotteisten mallien tuonti onnistuisi mahdollisimman helposti, saadaan ne ladattua käyttämällä erityisiä latausluokkia.

Loader-luokka on luokka tai oikeammin rajapinta, jonka toteuttava luokka osaa lukea muiden 3d-mallinnusohjelmien luomia kolmiulotteisia malleja ja muuntaa ne Java 3D:n tuntemaan muotoon.

Ote Loader-rajapinnan dokumentaatiosta:

> ”The Loader interface is used to specify the location and elements of a file format to load. The interface is used to give loaders of various file formats a common public interface” [37]

Java 3D -rajapinnan mukana tulevat valmiit esimerkkiluokat wavefront obj sekä lightwave lw 3d-mallinnustiedostojen käyttöön. Latausluokat löytyvät com.sun.j3d.loaders -paketista tai kolmansien osapuolien omista Java-paketeista. Esimerkiksi J3D.org sivustolta [38] löytyy lataus-luokkakirjastoja lähes jokaiselle tiedostoformaatille.

Seuraavana esimerkkinä käymme läpi Loader-luokan käyttöä. Käytämme hyväksi kaupallisen NewDawn Software yhtiön julkaisemaa Quake2 tietokonepelin MD2-mallitiedoston tiedostonlataaja-luokkakirjastoa. [39]

Toteutamme esimerkin siten, että meillä on erillinen ryhmähaara mallille, ja toinen ryhmähaara kaikelle muulle. Aluksi luomme erillisen metodin, joka lataa mallin käyttäen NewDawn md2loader -kirjaston tarjoamia luokkia ja palauttaa sitten meille ryhmähaara-olion, jossa tämä malli on ladattuna.

Mallinnustiedoston lataus on kohtuullisen yksinkertaista. Teemme vain uuden instanssin MD2Loader-luokasta ja käytämme sen loadWithPCX-metodia lataamaan tris.md2-mallitiedoston sekä teksturoimaan sen jungle.pcx-tiedoston perusteella. Lopuksi lisäämme sen väliaikaiseen ryhmähaaraan lapsisolmuksi ja palautamme luodun ryhmäsolmun.

```
private BranchGroup loadModel(String md2, String texture) {
    try {
        MD2Loader loader = new MD2Loader();
        MD2Model model = loader.loadWithPCX(
                new FileInput Stream(md2),
                new FileInputSream(texture));
        MD2ModelInstance jungle = model.getInstance();
        //asetaan animaatioksi juoksu
        jungle.setAnimation(”run”);
        BranchGroup temp = new  Branchgroup;
        temp.addChild(jungle);
        return temp;
    }
    catch (Exception ex){
        System.out.println(ex);
```

```
    }
    return null;
}
```

Seuraavaksi luomme pääohjelmassa käytettävät haarat: juuren, ryhmähaaran sekä muunnosryhmähaaran. Lisäämme munnosryhmähaaraan lapsisolmuksi loadModel-metodin palauttaman ryhmäsolmun. Lopuksi kokoamme maisemagraafin ja ajamme vielä compile-metodin pääryhmähaaralle.

```
public static void main(String args[]){
    SimpleUniverse juuri = new SimpleUniverse;
    BranchGroup bg = new BranchGroup();
    TransformGroup tg = new TransformGroup();
    tg.addChild(new BranchGroup(loadModel));
    bg.addChild(tg);
    bg.compile();
    juuri.addBranchGroup(bg);
    juuri.getViewingPlatform().setNominalViewingTransform();
}
```

Kun ajamme ohjelman, näemme juoksevan hahmon. Mikäli haluaisimme hahmon seisovan, voisimme asettaa animaation seisonta-asentoon kutsumalla setAnimation-metodia String ”stand” -parametrillä. Muita mahdollisia parametreja riippuen mallista ovat attack, jump ja death. Täydellinen lähdekoodi löytyy viitteestä [42].

Kuva 27 Quake2 mallitiedosto ladattuna

10 YHTEENVETO

Opinnäytetyön tarkoituksena on pyrkiä perehdyttämään lukija Java 3D -rajapinnan käytön perusteisiin sekä toimia ponnahduslautana Java 3D -rajapinnan edistyneenpään käyttöön. Java 3D -rajapinta tarjoaa Java-ohjelmointikieleen keinot kolmiulotteisen avaruuden hallitsemiseksi.

Java 3D -rajapinnan käyttämä hierarkinen maisemagraafi-arkkitehtuuri mahdollistaa kolmiulotteisen avaruuden kappaleiden sekä niille tehtävien muunnosten hallinnan joustavalla ja helpolla tavalla. Näin olemassa olevaan Java-ohjelmaan voidaan helposti lisätä kolmiulotteistagrafiikkaa. Java 3D -rajapintaa käyttävien ohjelmien etuna on myös, että ne toimivat kaikkialla missä Java-virtuaalikone sekä Java 3D ovat käytössä, riippumatta alla olevasta käyttöjärjestelmästä tai laitteistosta.

VIITELUETTELO

[1] Java 3D [http://en.wikipedia.org/wiki/Java_3D#History , Viitattu 30.03.2008]

[2] Java 3D Users

[http://wiki.java.net/bin/view/Javadesktop/Java3DUsers , Viitattu 30.03.2008]

[3] 3D User Interfaces with Java 3D, Jon Barrilleaux, 2001. Kappale 10.1.

[4] SceneGraph Overview

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/doc/files/SceneGraphOverview.html , Viitattu 30.03.2008]

[5] HelloWorld

[http://www.pahvilaatikko.org/j3d/src/esim1/HelloWorld.java, Viitattu 30.03.2008]

[6] Simple Universe

[http://download.java.net/media/java3d/javadoc/1.5.1/com/sun/j3d/utils/universe/SimpleUniverse.html ,Viitattu 30.03.2008]

[7] Locale

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/Locale.html , Viitattu 30.03.2008]

[8] BranchGroup

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/BranchGroup.html ,Viitattu 30.03.2008]

[9] TransformGroup

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/TransformGroup.html ,Viitattu 30.03.2008]

[10] Transform3D

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/Transform3D.html ,Viitattu 30.03.2008]

[10] HelloWorld2

[http://www.pahvilaatikko.org/j3d/src/esim2/HelloWorld2.java ,Viitattu 30.03.2008]

[11] Vector3f

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/vecmath/Vector3f.html ,Viitattu 30.03.2008]

[12] HelloWorld3

[http://www.pahvilaatikko.org/j3d/src/esim3/HelloWorld3.java , Viitattu 30.03.2008]

[13] Appendix Equations

[http://java.sun.com/products/java-media/3D/forDevelopers/J3D_1_3_API/j3dguide/AppendixEquations.html , Viitattu 2.1.2008]

[14] Appearance

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/Appearance.html ,Viitattu 30.03.2008]

[15] Material

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/Material.html ,Viitattu 30.03.2008]

[16] DirectionalLight

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/DirectionalLight.html ,Viitattu 30.03.2008]

[17] Valot

[http://www.pahvilaatikko.org/j3d/src/esim4/Valot.java ,Viitattu 30.03.2008]

[18] Varjo

[http://www.pahvilaatikko.org/j3d/src/esim5/Varjo.java ,Viitattu 30.03.2008]

[19] Alpha

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/Alpha.html , Viitattu 30.03.2008]

[20] RotationInterpolator

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/RotationInterpolator.html , Viitattu 30.03.2008]

[18] RotationInterpolator esimerkki

[http://www.pahvilaatikko.org/j3d/src/Animaatiot/Animaatio1Pyorittaja.java Viitattu, 30.03.2008]

[22] PositionInterpolator

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/PositionInterpolator.html , Viitattu 30.03.2008]

[23] PositionInterpolator esimerkki

[http://www.pahvilaatikko.org/j3d/src/Animaatiot/Animaatio2Liikuttaja.java , Viitattu 30.03.2008]

[24] PositionPathInterpolator

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/PositionPathInterpolator.html , Viitattu 30.03.2008]

[25] PositionPathInterpolator esimerkki

[http://www.pahvilaatikko.org/j3d/src/Animaatiot/Animaatio3PosPath.java , Viitattu 30.03.2008]

[26] RotationPositionPathInterplator

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/RotPosPathInterpolator.html , Viitattu 30.03.2008]

[27] RotPosPathInterplator

[http://www.pahvilaatikko.org/j3d/src/Animaatiot/Animaatio4RotPosPath.java , Viitattu 30.03.2008]

[28] Hiiri

[http://www.pahvilaatikko.org/j3d/src/Interaktio/HiiriInteraktio.java, Viitattu 30.03.2008]

[29] Näppäimistö

[http://www.pahvilaatikko.org/j3d/src/Interaktio/HiiriNappaimistoInteraktio.java , Viitattu 30.03.2008]

[30] Pallot

[http://www.pahvilaatikko.org/j3d/src/Teksturointi/Tekstuuri1.java , Viitattu 30.03.2008]

[31] Mipmap

[http://www.pahvilaatikko.org/j3d/src/Teksturointi/Mipmap.java , Viitattu 30.03.2008]

[32] Mainostaulu

[http://www.pahvilaatikko.org/j3d/src/Teksturointi/Mainostaulu.java , Viitattu 30.03.2008]

[33] DistanceLOD

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/DistanceLOD.html , Viitattu 30.03.2008]

[34] Switch

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/Switch.html , Viitattu 30.03.2008]

[35] LOD esimerkki

[http://www.pahvilaatikko.org/j3d/src/Teksturointi/DistanceLOD.java , Viitattu 30.03.2008]

[36] Shape3D

[http://download.java.net/media/java3d/javadoc/1.5.1/javax/media/j3d/Shape3D.html , Viitattu 30.03.2008]

[37] Loaders

[http://download.java.net/media/java3d/javadoc/1.5.1/com/sun/j3d/loaders/Loader.html , Viitattu 30.03.2008]

[38] J3D.org [http://www.j3d.org/ , Viitattu 31.03.2008]

[39] NewDawn Software [http://www.newdawnsoftware.com/ , Viitattu 31.03.2008]

[40] Trilinear Interplator

[http://en.wikipedia.org/wiki/Trilinear_interpolation , Viitattu 31.03.2008]

[41] Compile-method

[http://java3d.j3d.org/tutorials/quick_fix/compile.html , Viitattu 31.03.2008]

[42] MD2-lataaja esimerkki

[http://www.pahvilaatikko.org/j3d/src/Shape3D/MD2Test.java , Viitattu 30.03.2008]